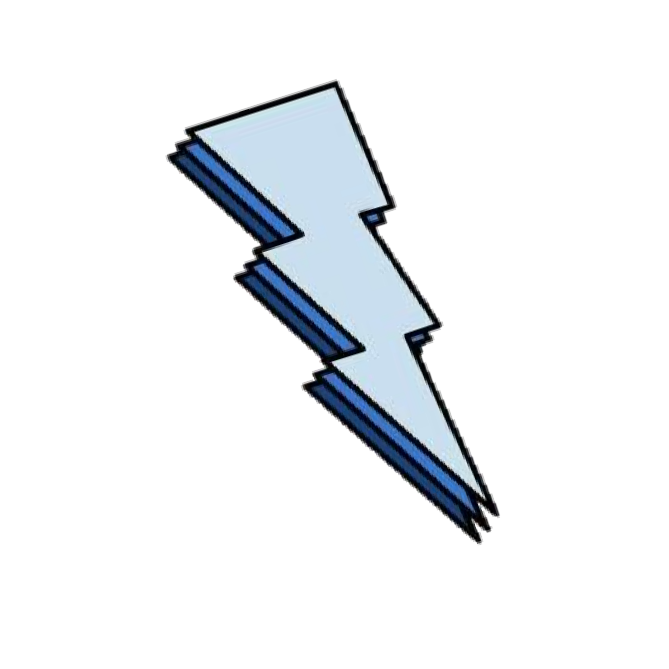
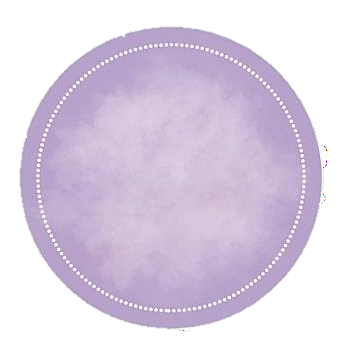
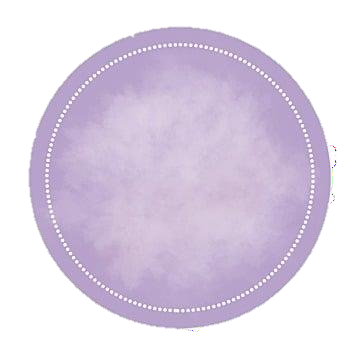
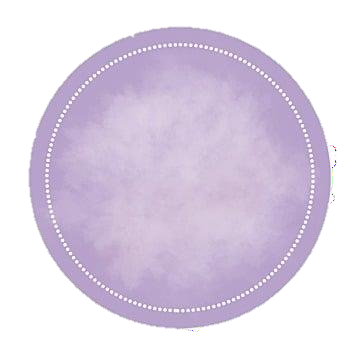
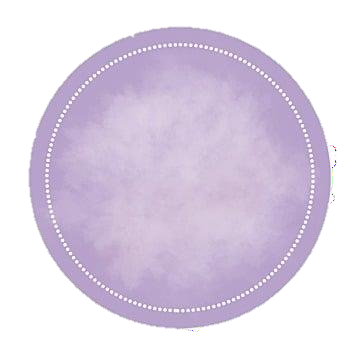
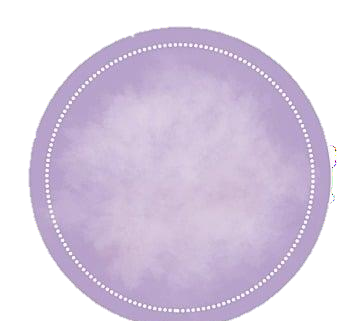


INSTITUTO POLITECTO NACIONAL Escuela SUPERIOR DE CÓMPUTO



Profesor:

Cortes Galicia Jorge



Unidad de Aprendizaje: Sistemas Operativos





Introducción

Esta práctica tiene como propósito principal que nos familiaricemos con las interfaces de llamadas al sistema de dos sistemas operativos fundamentales: Linux y Windows. A lo largo de este ejercicio, exploraremos las diferencias y similitudes en las llamadas al sistema de ambos entornos, utilizando el lenguaje de programación C como nuestra herramienta principal.

Iniciaremos con tareas básicas en el entorno Windows, tales como la creación y manipulación de archivos. Posteriormente, nos adentraremos en el entorno Linux, donde exploraremos aspectos como la montura de dispositivos USB, la edición de archivos utilizando el editor de texto `gedit`, y la consulta de llamadas al sistema a través del manual en línea (`man`).

Nos enfocaremos en comprender la funcionalidad de una variedad de llamadas al sistema tanto en Linux como en Windows. Investigaremos su propósito y aprenderemos cómo utilizarlas en la programación. A través de la implementación de programas en C, pondremos en práctica los conocimientos adquiridos, creando archivos, modificando permisos, listando información de archivos y realizando otras operaciones básicas tanto en entornos Linux como en Windows. Este ejercicio nos proporcionará una base sólida para entender el funcionamiento interno de los sistemas operativos y nos equipará con habilidades esenciales para la programación y administración de sistemas en diversos entornos.

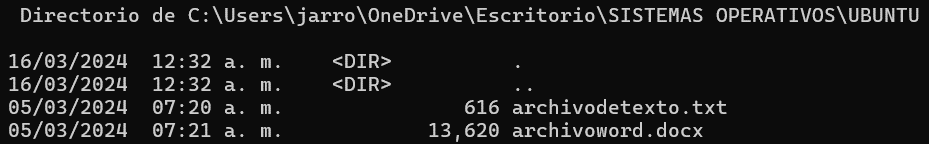
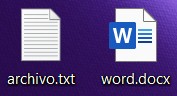
Finalmente, esta práctica nos brindará la oportunidad de comprender cómo se manejan las llamadas al sistema en dos de los sistemas operativos más utilizados en el ámbito profesional. Al explorar las diferencias entre Linux y Windows en términos de llamadas al sistema, ampliaremos nuestra comprensión sobre cómo interactúan los programas con el núcleo del sistema operativo para llevar a cabo diversas tareas, desde operaciones básicas de entrada y salida hasta la gestión avanzada de archivos y recursos del sistema. Este conocimiento no solo nos permitirá adentrarnos en la programación de bajo nivel, sino que también nos dotará de herramientas para desarrollar soluciones más eficientes y versátiles en el ámbito de la informática.





Desarrollo

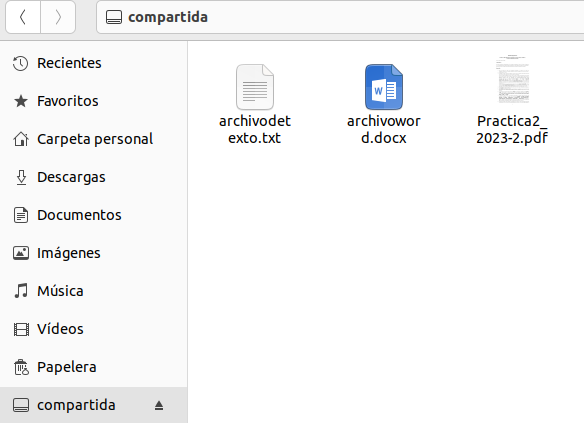
1. Cree un archivo de texto (con cualquier contenido) y un archivo en Word (con cualquier contenido) en el sistema operativo Windows y guárdelo en una memoria usb.



1. Inicie sesión en Linux.
2. Verifique si está montada la unidad de memoria usb en su sistema, para ello introduzca una memoria usb y observe si es reconocida en el escritorio. Nota: En algunas distribuciones de Linux no está montada automáticamente la unidad de memoria usb, sino está montada la unidad usb, móntela mediante la terminal de la siguiente manera: Cree un directorio en

/media/Nombre\_Memoria, donde Nombre\_Memoria es el nombre que quiera darle a su directorio. Introduzca una memoria usb en la unidad. Escriba: mount -t vfat /dev/sdX

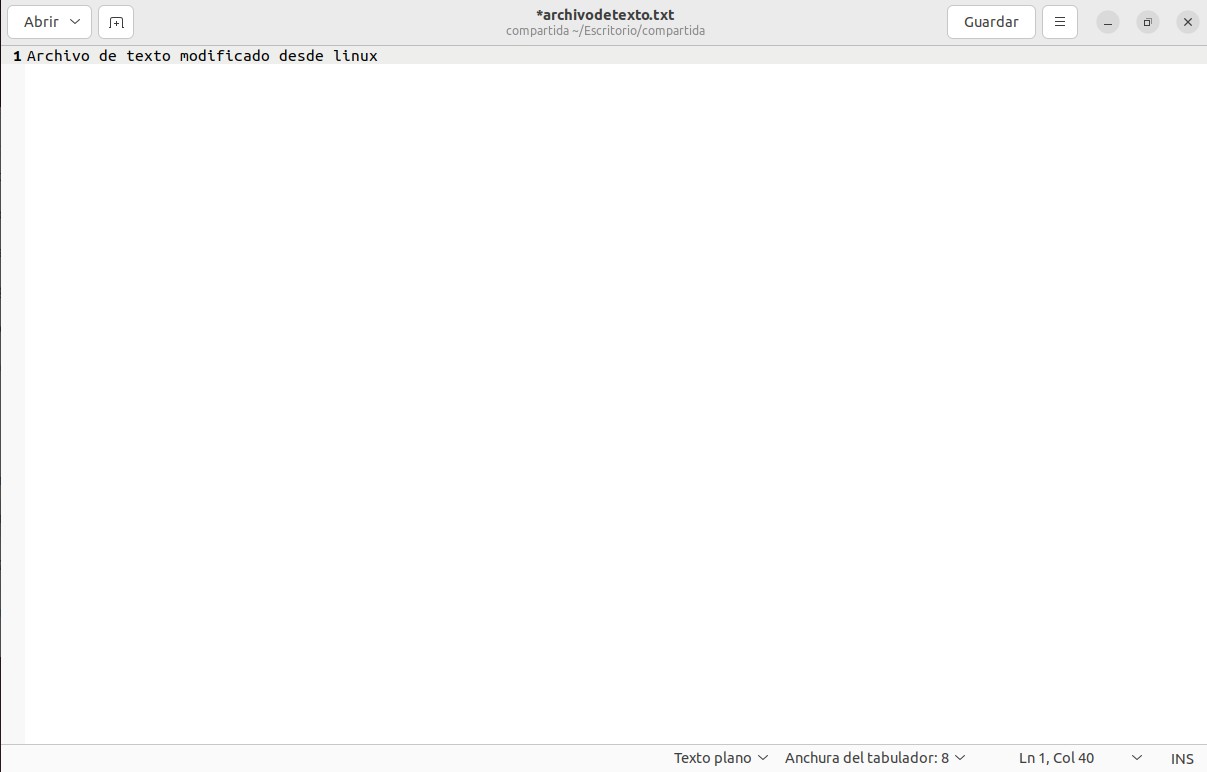
/media/Nombre\_Memoria, donde en sdX la X representa el número que el sistema le asigna a su memoria usb. Escriba cd /media/Nombre\_Memoria, si el cambio de directorio es realizado, su unidad de usb ha sido montada con éxito y puede continuar en el punto 4. Puede desmontar su memoria usb de la siguiente manera: umount /media/sd0. Utilice el comando cd /media/Nombre\_Memoria para comprobar que realmente se desmontó la unidad. Tanto montaje y desmontaje requieren permisos de súper usuario (root).

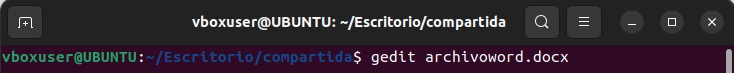


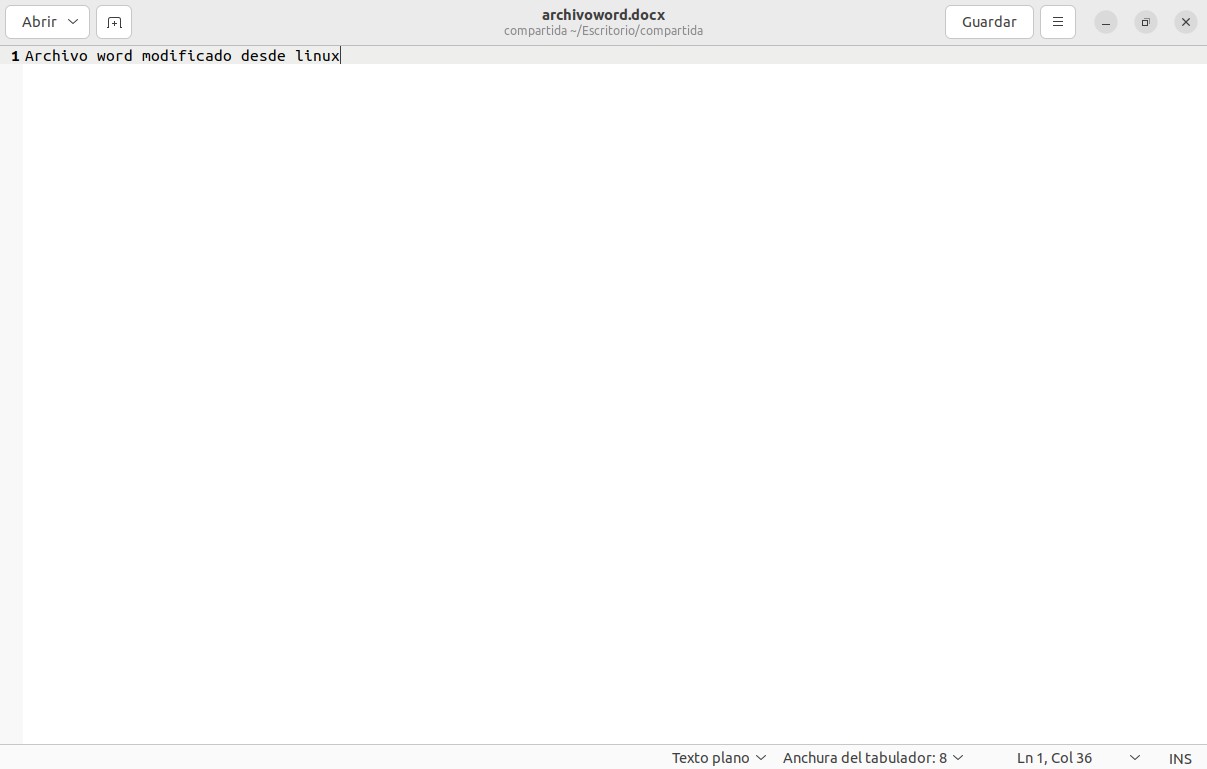
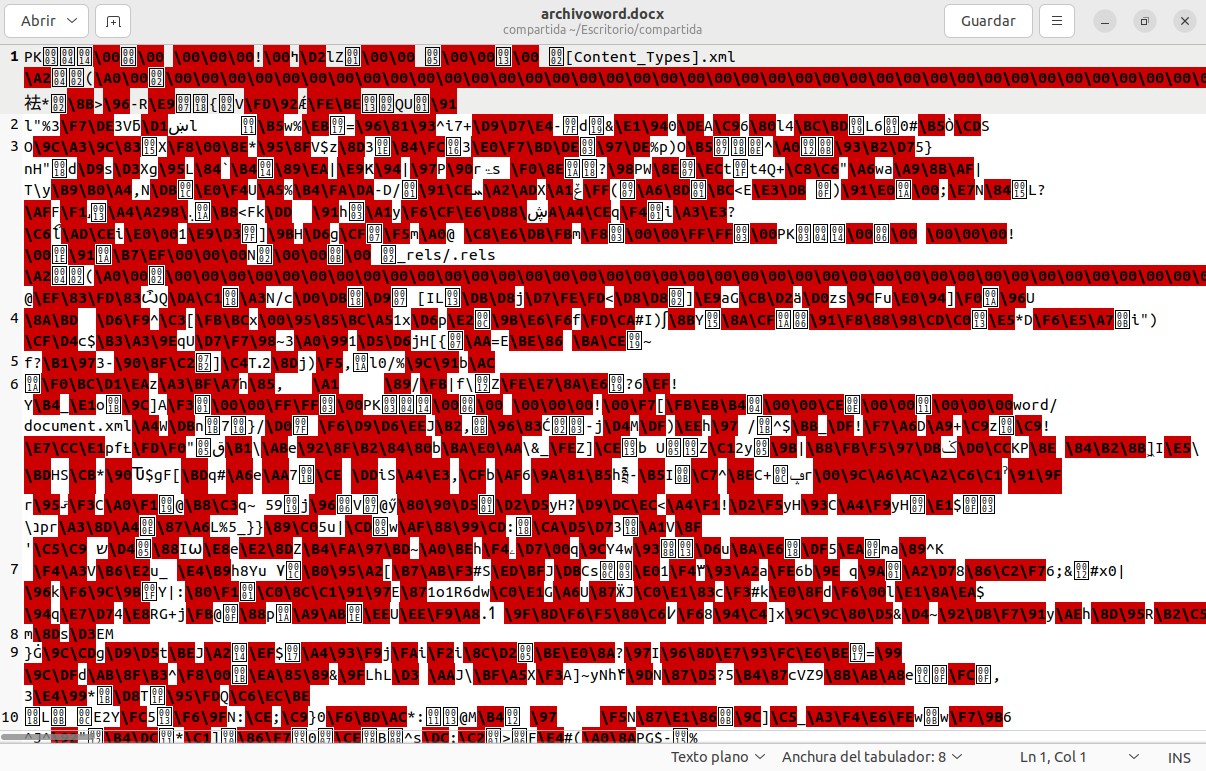
1. Edite tanto el contenido del archivo de texto como de Word, modificándolo mediante el uso de gedit. Guarde sus archivos.





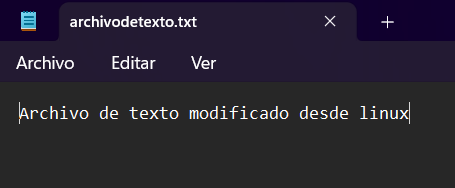


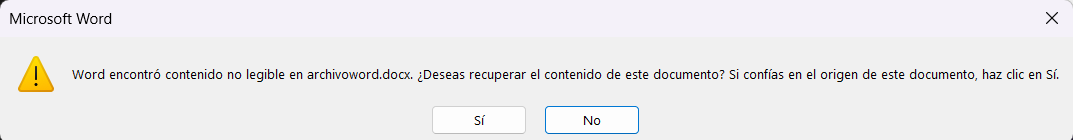


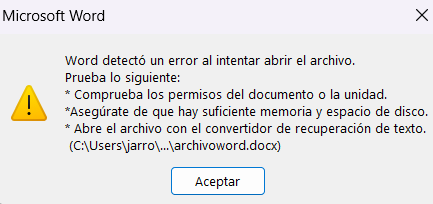


1. Inicie sesión en Windows y observe el contenido de sus archivos en su memoria usb. ¿Se observan las modificaciones realizas en Linux?, explique el por qué si o no se observan.









Cuando observamos el contenido de los archivos en una memoria USB en Windows después de realizar modificaciones en Linux, notamos que las modificaciones realizadas en el archivo de texto son visibles, mientras que el archivo de Word parece estar corrupto. Esta discrepancia se debe a las diferencias en la forma en que se gestionan los archivos de texto y los archivos binarios en diferentes sistemas operativos, así como a la compatibilidad de formatos entre Linux y Windows.

En el caso del archivo de texto, la interoperabilidad entre sistemas operativos es mayor, ya que estos archivos generalmente están formateados utilizando codificaciones de caracteres estándar, como UTF-8, y no tienen estructuras internas complejas. Por lo tanto, las modificaciones realizadas en un archivo de texto en Linux probablemente se verán correctamente en Windows, ya que ambos sistemas operativos pueden interpretar el contenido de manera similar.

Por otro lado, los archivos de Word (como los archivos .docx) son archivos binarios que siguen las especificaciones de formato de Microsoft Office. Estos archivos pueden contener datos estructurados y formatos específicos que pueden no ser compatibles o interpretados correctamente por aplicaciones diferentes a Microsoft Word. En consecuencia, cuando se modifican estos archivos en Linux y luego se abren en Windows, es posible que se corrompan debido a diferencias en la interpretación del formato o a la pérdida de datos durante la transferencia entre sistemas operativos.

1. A través de su manual en línea de Linux (man) en la segunda o tercera sección (man 2 o man 3), investigue y reporte para qué sirven las siguientes llamadas al sistema de Linux: open, close, read, write, creat, lseek, access, stat, chmod, chown, fcntl, chdir, mkdir, opendir, y readdir.

# oPen:

Propósito: Abre un archivo para lectura, escritura o ambas. Si el archivo no existe, puede crearlo dependiendo de las banderas especificadas.

Sintaxis: fd = open (pathname, flags, modes); Parámetros:

* pathname: nombre del archivo.
* flags: modo de apertura, para lectura, escritura, etc.
* modes: permisos del archivo en caso de que se deba crear.
* fd: entero que representa el descriptor del archivo.

# close:

Propósito: Cierra un descriptor de archivo previamente abierto para que sea reusado

Sintaxis: close(fd)

parámetros: fd es el descriptor del archivo.

# read:

Propósito: Lee datos desde un archivo abierto y los coloca en un búfer. Sintaxis: number = read (fd, buffer, count);

Parametros:

fd: descriptor de archivo que devuelve la llamada open

buffer: dirección del buffer donde se van a colocar los datos leídos count: número de bytes que el usuario quiere leer

number: número de bytes leídos si es positivo en caso contrario la ejecución no ha sido correcta

# write:

Propósito: Escribe datos desde un búfer a un archivo abierto Sintaxis: number=write (fd, buffer, count);

Parametros:

fd: descriptor de archivo que devuelve la llamada open

buffer: dirección del buffer donde se van a colocar los datos leídos count: número de bytes que el usuario quiere leer

number: número de bytes leídos si es positivo en caso contrario la ejecución no ha sido correcta

# creat:

Propósito: Se usa para crear un archivo en la ruta especificada aunque normalmente, se usa open en su lugar.

Sintaxis: int creat(const char \*pathname, mode\_t mode); Parametros:

* pathname: Es una cadena de caracteres que representa la ruta y el nombre del archivo que se desea crear o truncar.
* mode: Es un entero sin signo de tipo mode\_t que especifica los permisos para el archivo que se va a crear. Los permisos están sujetos a la máscara de permisos umask del proceso que realiza la llamada.



# lseek:

Proposito: Cambia la posición del puntero de lectura/escritura dentro de un archivo Sintaxis: off\_t lseek(int fd, off\_t offset, int whence);

Parametros:

* fd: Es el descriptor de archivo sobre el cual se va a realizar la operación de posicionamiento. Este descriptor de archivo es devuelto por la llamada al sistema open() o por otra llamada al sistema que abra o cree un archivo.
* offset: Es un valor de tipo off\_t, que es un tipo entero que se utiliza para representar desplazamientos en archivos. Este valor indica la cantidad de bytes que se debe desplazar el puntero de lectura/escritura. Puede ser positivo (para desplazarse hacia adelante) o negativo (para desplazarse hacia atrás).
* whence: Es un entero que especifica la referencia para el desplazamiento. Puede tener uno de los siguientes valores:

SEEK\_SET: El desplazamiento se realiza desde el inicio del archivo.

SEEK\_CUR: El desplazamiento se realiza desde la posición actual del puntero de lectura/escritura. SEEK\_END: El desplazamiento se realiza desde el final del archivo.

# access:

Proposito: Comprueba si el proceso tiene permiso para acceder a un archivo o directorio especificado.

Sintaxis: int access(const char \*pathname, int mode); Parametros:

* pathname: Es una cadena de caracteres que representa la ruta y el nombre del archivo o directorio que se desea verificar.
* mode: Es un entero que especifica el tipo de acceso que se desea verificar. Puede ser una combinación de las siguientes constantes:

F\_OK: Verifica si el archivo o directorio existe.

R\_OK: Verifica si el proceso tiene permisos de lectura en el archivo o directorio. W\_OK: Verifica si el proceso tiene permisos de escritura en el archivo o directorio. X\_OK: Verifica si el proceso tiene permisos de ejecución en el archivo o directorio.

# Patrón de fondo Descripción generada automáticamente con confianza mediastat:

Propósito: Devuelve información sobre un archivo, como tamaño, propietario, permisos, etc. Sintaxis: int stat(const char \*pathname, struct stat \*statbuf);

Parametros:

* pathname: Es una cadena de caracteres que representa la ruta y el nombre del archivo o directorio del cual se desea obtener información.
* statbuf: Es un puntero a una estructura stat donde se almacenarán los atributos del archivo o directorio.

La estructura stat es definida por el sistema y contiene varios campos que representan los atributos del archivo o directorio. Algunos de los campos comunes en la estructura stat incluyen:

st\_mode: Los permisos del archivo.

st\_uid: El identificador de usuario del propietario del archivo. st\_gid: El identificador de grupo del propietario del archivo. st\_size: El tamaño del archivo en bytes.

st\_atime: La fecha y hora del último acceso al archivo. st\_mtime: La fecha y hora de la última modificación del archivo.

st\_ctime: La fecha y hora del último cambio de estado del archivo.

# chmod:

Propósito: Cambia los permisos de un archivo.

Sintaxis: int chmod(const char \*pathname, mode\_t mode); Parametros:

* pathname: Es una cadena de caracteres que representa la ruta y el nombre del archivo o directorio cuyos permisos se desean modificar.
* mode: Es un entero sin signo de tipo mode\_t que especifica los nuevos permisos que se desean establecer para el archivo o directorio. Los permisos se especifican utilizando las constantes definidas en la biblioteca sys/stat.h. Las constantes comunes para los permisos incluyen:

S\_IRUSR: Permiso de lectura para el propietario del archivo. S\_IWUSR: Permiso de escritura para el propietario del archivo.

S\_IXUSR: Permiso de ejecución para el propietario del archivo. S\_IRGRP: Permiso de lectura para el grupo del propietario del archivo.

S\_IWGRP: Permiso de escritura para el grupo del propietario del archivo. S\_IXGRP: Permiso de ejecución para el grupo del propietario del archivo. S\_IROTH: Permiso de lectura para otros usuarios.

S\_IWOTH: Permiso de escritura para otros usuarios. S\_IXOTH: Permiso de ejecución para otros usuarios.

# chown:

Propósito: Cambia el propietario y/o el grupo de un archivo.

Sintaxis: int chown(const char \*pathname, uid\_t owner, gid\_t group); Parametros:

* pathname: Es una cadena de caracteres que representa la ruta y el nombre del archivo o directorio cuyo propietario y grupo se desean modificar.
* owner: Es un valor de tipo uid\_t que especifica el nuevo identificador de usuario (UID) que se desea establecer como propietario del archivo o directorio.
* group: Es un valor de tipo gid\_t que especifica el nuevo identificador de grupo (GID) que se desea establecer como grupo del archivo o directorio.

# fcntl:

Propósito: Realiza varias operaciones de control en un descriptor de archivo, como establecer banderas de archivo.

Sintaxis: int fcntl(int fd, int cmd, ... /\* arg \*/ ); Parametros:

* fd: Es el descriptor de archivo sobre el cual se realizará la operación de control.
* cmd: Es un entero que especifica la operación que se realizará. Puede ser una de las siguientes constantes definidas en <fcntl.h> o una operación específica del sistema.
* arg: Es un argumento adicional que puede ser necesario dependiendo del valor de cmd. Algunas operaciones comunes que se pueden realizar con fcntl() incluyen:

Duplicación de descriptores de archivo (F\_DUPFD, F\_DUPFD\_CLOEXEC).

Obtención o ajuste de banderas de archivo (F\_GETFD, F\_SETFD, F\_GETFL, F\_SETFL).

Bloqueo de archivos (F\_SETLK, F\_SETLKW, F\_GETLK).

Obtención o ajuste del tamaño del descriptor de archivo (F\_GETOWN, F\_SETOWN).

# chdir:

Propósito: Cambia el directorio de trabajo actual del proceso Sintaxis: int chdir(const char \*path);

Parametros:

path: Es una cadena de caracteres que representa la ruta al directorio al que se desea cambiar.

# mkdir:

Proposito: Crea un nuevo directorio

Sintaxis: int mkdir(const char \*pathname, mode\_t mode); Parametros:

* pathname: Es una cadena de caracteres que representa la ruta y el nombre del directorio que se desea crear.
* mode: Es un entero sin signo de tipo mode\_t que especifica los permisos para el nuevo directorio. Los permisos están sujetos a la máscara de permisos umask del proceso que realiza la llamada.

# oPendir:

Proposito: Abre un directorio para su posterior lectura Sintaxis: DIR \*opendir(const char \*name); Parámetros:

name: Es una cadena de caracteres que representa la ruta al directorio que se desea abrir.

# readdir:

Proposito: Lee la siguiente entrada de un directorio abierto previamente Sintaxis: struct dirent \*readdir(DIR \*dirp);

Parametros:

dirp: Es un puntero al directorio abierto, obtenido previamente mediante la función opendir().

1. A través del sitio MSDN de Microsoft, investigue y reporte para qué sirven las siguientes llamadas al sistema de Windows: OpenFile, CloseFile, ReadFile, WriteFile, CreateFile, SetFilePointer, CreateDirectory, SetCurrentDirectory, GetFileAttributes, SetFileAttributes, FindFirstFile, y FindNextFile. Investigue y reporte si existe una llamada idéntica a chmod en Windows, en caso de no existir una llamada idéntica, investigar que llamada equivalente existe en Windows.

# OPenFile:

Propósito: La función OpenFile crea, abre, reabre o elimina un archivo.

Nota: Esta función tiene capacidades limitadas y no se recomienda para el desarrollo de nuevas aplicaciones. En su lugar, se sugiere usar la función CreateFile.

Sintaxis: HFILE OpenFile(LPCSTR lpFileName, LPOFSTRUCT lpReOpenBuff, UINT uStyle) Parámetros:

lpFileName: La ruta del archivo que se va a abrir.

lpReOpenBuff: Un puntero a una estructura OFSTRUCT para recibir información sobre el archivo abierto.

uStyle: Un conjunto de indicadores que especifican cómo se debe abrir el archivo.

# CloseFile:

Propósito: La función CloseHandle cierra un identificador de objeto abierto (como un archivo, dispositivo de comunicaciones, tubería, etc.).

Se utiliza para liberar recursos y finalizar el acceso a un objeto. Sintaxis: BOOL CloseHandle(HANDLE hObject)

Parámetros:

hObject: El identificador del archivo o del objeto que se va a cerrar.

# ReadFile:

Propósito: La función ReadFile lee datos desde un archivo o dispositivo de entrada/salida (E/S). Puede realizarse de manera síncrona o asíncrona.

Sintaxis: BOOL ReadFile(HANDLE hFile, LPVOID lpBuffer, DWORD nNumberOfBytesToRead, LPDWORD lpNumberOfBytesRead, LPOVERLAPPED lpOverlapped)

Parámetros:

hFile: El identificador del archivo del que se van a leer los datos. lpBuffer: Un búfer para almacenar los datos leídos.

nNumberOfBytesToRead: El número de bytes que se van a leer del archivo. lpNumberOfBytesRead: Un puntero a una variable para almacenar el número de bytes que se

han leído.

lpOverlapped: Un puntero a una estructura OVERLAPPED opcional para operaciones de E/S superpuestas.

# WriteFile:

Propósito: La función WriteFile escribe datos en un archivo o dispositivo de E/S. Al igual que ReadFile, puede utilizarse de manera síncrona o asíncrona.

Sintaxis: BOOL WriteFile(HANDLE hFile, LPCVOID lpBuffer, DWORD nNumberOfBytesToWrite, LPDWORD lpNumberOfBytesWritten, LPOVERLAPPED lpOverlapped)

Parámetros:

Igual que ReadFile, pero para escritura.

# CreateFile:

Propósito: La función CreateFile crea o abre un archivo o dispositivo de E/S.

Se utiliza para obtener un identificador que permite acceder al archivo o dispositivo para operaciones de lectura, escritura, etc.

Sintaxis: HANDLE CreateFile(LPCTSTR lpFileName, DWORD dwDesiredAccess, DWORD dwShareMode, LPSECURITY\_ATTRIBUTES lpSecurityAttributes, DWORD dwCreationDisposition, DWORD dwFlagsAndAttributes, HANDLE hTemplateFile)

Parámetros:

Varios parámetros para especificar cómo se debe abrir o crear el archivo, como el nombre del archivo, el acceso deseado, el modo de uso compartido, entre otros.

# SetFilePointer:

Propósito: Sirve para establecer la posición actual del puntero de archivo para un archivo abierto.

Sintaxis: DWORD SetFilePointer(HANDLE hFile, LONG lDistanceToMove, PLONG lpDistanceToMoveHigh, DWORD dwMoveMethod)

Parámetros:

hFile: El identificador del archivo cuyo puntero de archivo se va a establecer. lDistanceToMove: El número de bytes que se moverá el puntero de archivo.

lpDistanceToMoveHigh: Puntero opcional a la parte alta de un valor de 64 bits que especifica la distancia a mover.

dwMoveMethod: Especifica cómo se interpreta el parámetro lDistanceToMove.

# CreateDirectory:

Propósito: Se utiliza para crear un nuevo directorio.

Sintaxis: BOOL CreateDirectory(LPCTSTR lpPathName, LPSECURITY\_ATTRIBUTES lpSecurityAttributes)

Parámetros:

lpPathName: La ruta del directorio que se va a crear.

lpSecurityAttributes: Un puntero a una estructura SECURITY\_ATTRIBUTES opcional que especifica los atributos de seguridad del nuevo directorio.

# SetCurrentDirectory:

Propósito: Se utiliza para cambiar el directorio actual del proceso. Sintaxis: BOOL SetCurrentDirectory(LPCTSTR lpPathName) Parámetros:

lpPathName: La ruta del nuevo directorio que se va a establecer como directorio actual.

# GetFileAttributes:

Propósito: Se utiliza para obtener los atributos de un archivo o directorio específico. Sintaxis: DWORD GetFileAttributes(LPCTSTR lpFileName)

Parámetros:

lpFileName: La ruta del archivo o directorio del que se van a obtener los atributos.

# SetFileAttributes:

Propósito: Sirve para establecer los atributos de un archivo o directorio específico. Sintaxis: BOOL SetFileAttributes(LPCTSTR lpFileName, DWORD dwFileAttributes) Parámetros:

lpFileName: La ruta del archivo o directorio cuyos atributos se van a establecer. dwFileAttributes: Los nuevos atributos que se van a establecer.

# FindFirstFile:

Propósito: Se utiliza para encontrar el primer archivo o directorio que coincida con un patrón de búsqueda especificado.

Sintaxis: HANDLE FindFirstFile(LPCTSTR lpFileName, LPWIN32\_FIND\_DATA lpFindFileData)

Parámetros:

lpFileName: El patrón de búsqueda utilizado para encontrar archivos o directorios. lpFindFileData: Un puntero a una estructura WIN32\_FIND\_DATA para recibir información

sobre el primer archivo o directorio encontrado.

# FindNextFile:

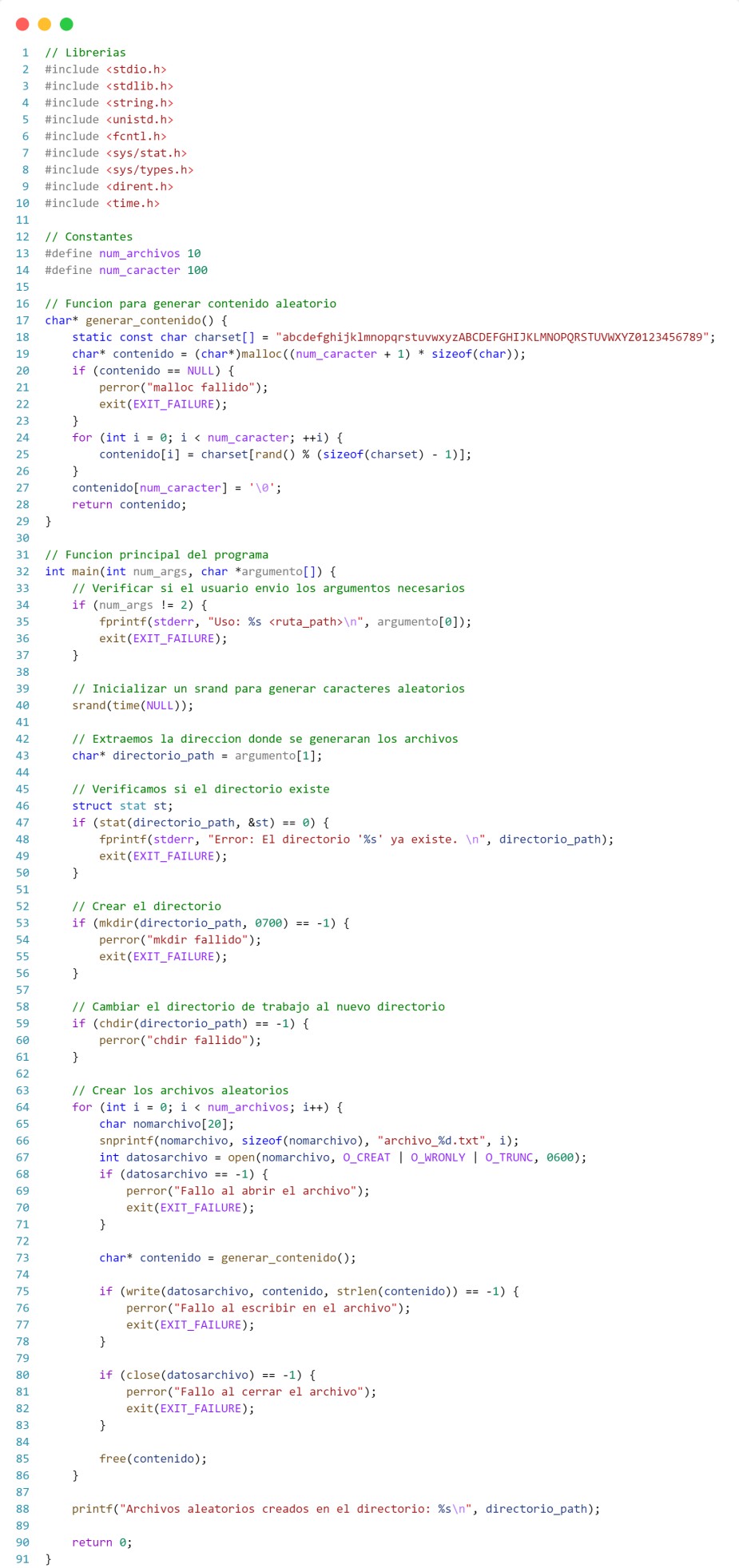
Propósito: Se utiliza para encontrar el siguiente archivo o directorio que coincida con un patrón de búsqueda después de una llamada exitosa a FindFirstFile.

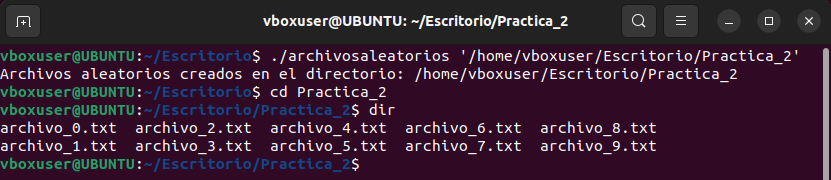
Sintaxis: BOOL FindNextFile(HANDLE hFindFile, LPWIN32\_FIND\_DATA lpFindFileData) Parámetros:

hFindFile: El identificador de búsqueda devuelto por FindFirstFile.

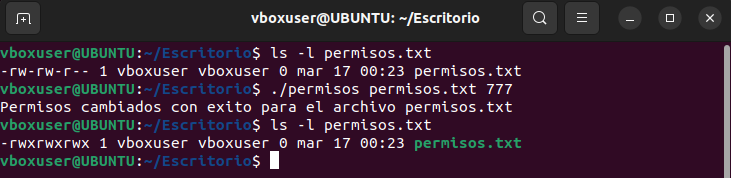
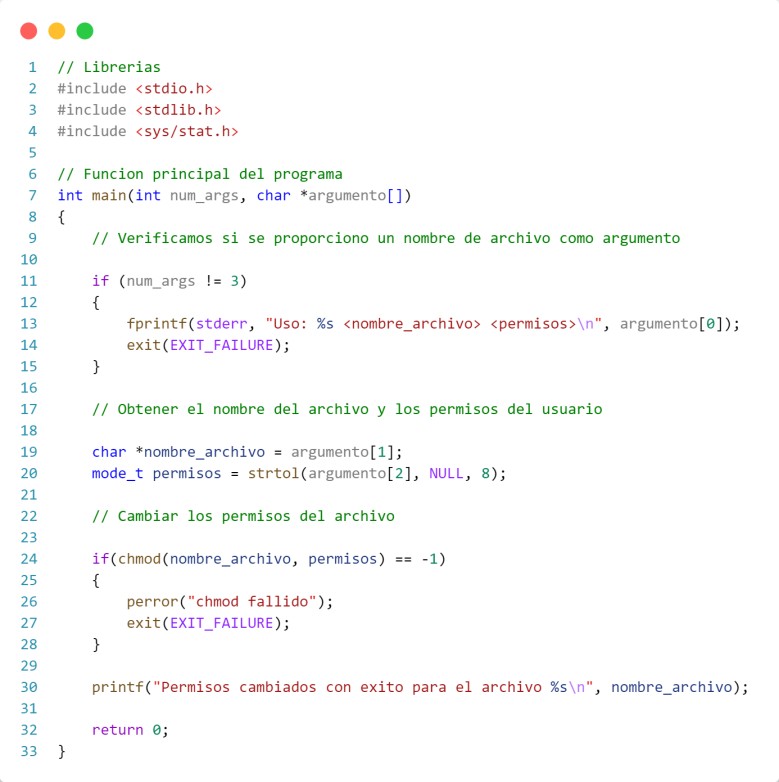
lpFindFileData: Un puntero a una estructura WIN32\_FIND\_DATA para recibir información sobre el siguiente archivo o directorio encontrado.

1. Utilizando las llamadas al sistema revisadas para Linux que sean necesarias, desarrolle un programa en C que cree una serie aleatoria de archivos (en una ruta especificada a 2 través de la línea de comando, el directorio no debe existir previamente), el contenido de los archivos serán cadenas que estén almacenadas en un arreglo dentro del programa.

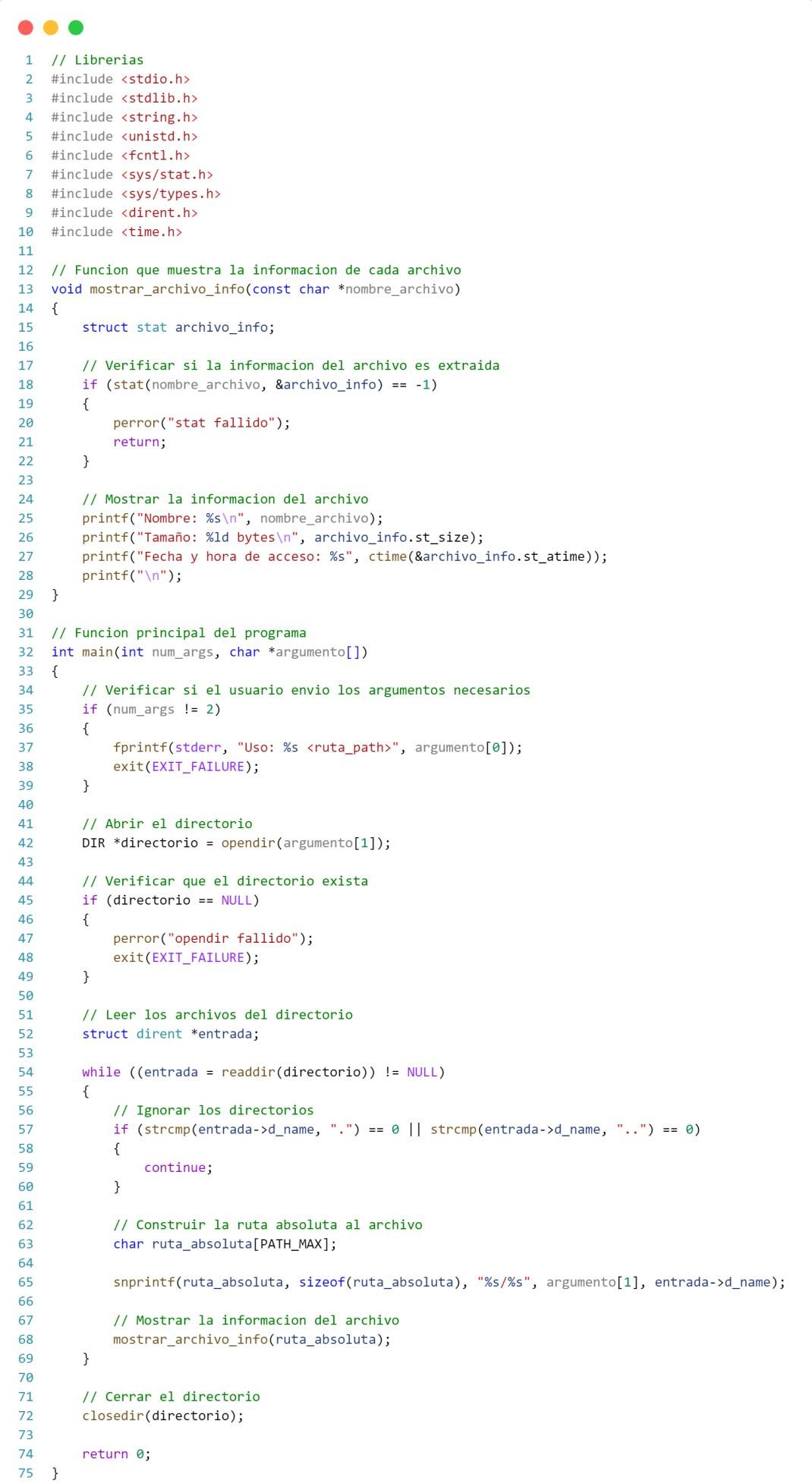




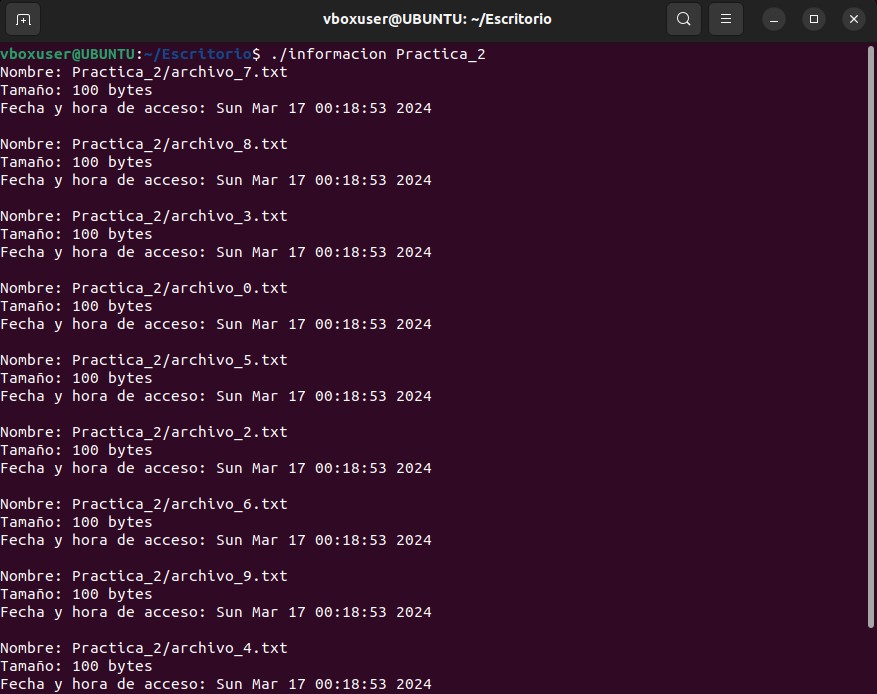
1. Una vez creados los archivos con sus contenidos por el programa del punto 8, y utilizando las llamadas al sistema revisadas para Linux que sean necesarias, desarrolle un programa en C para cambiar los permisos de un archivo seleccionado por el usuario.



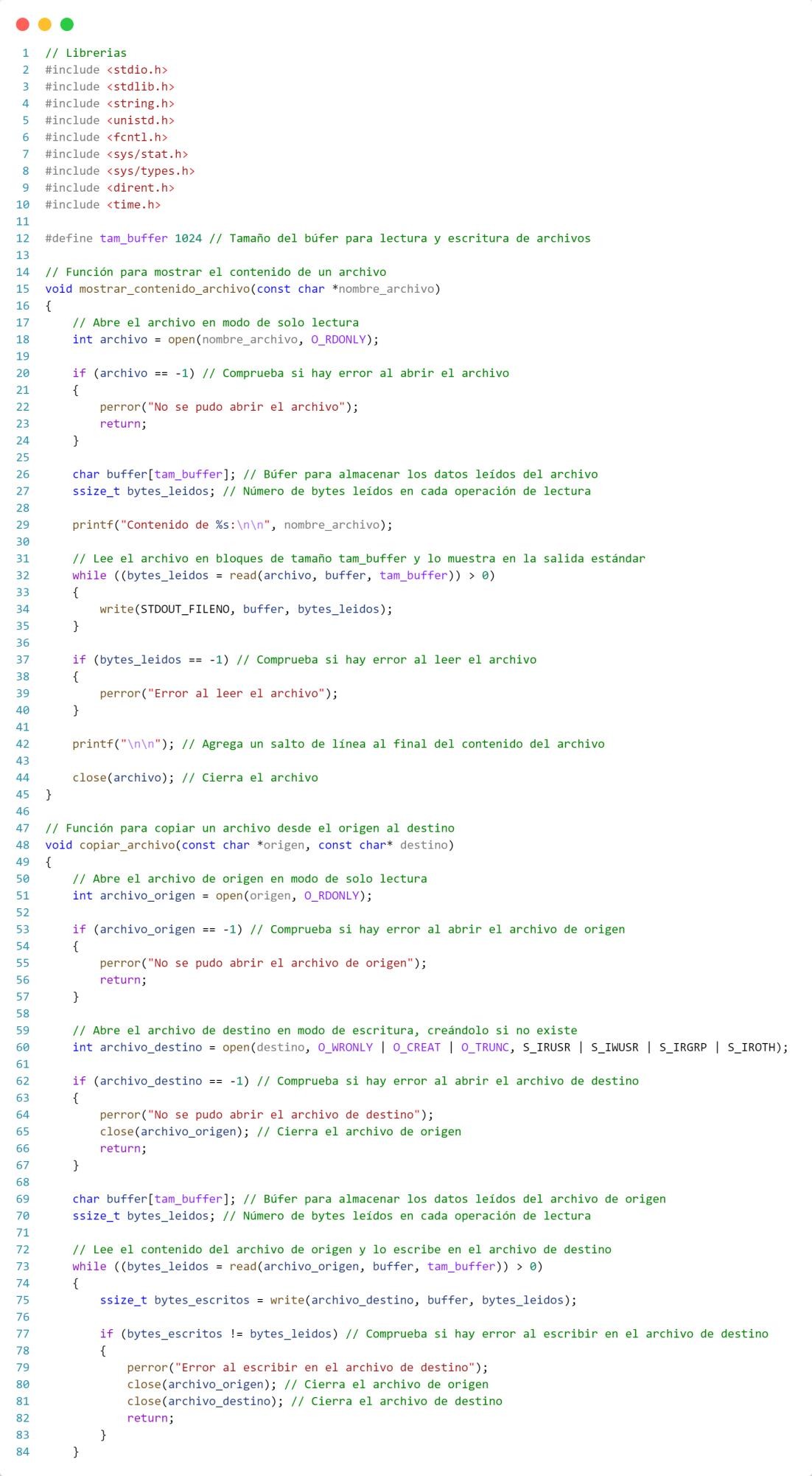
1. Una vez creados los archivos con sus contenidos por el programa del punto 8. y utilizando las llamadas al sistema revisadas para Linux que sean necesarias, desarrolle un programa en C que liste los archivos creados, mostrando su tamaño, fecha y hora de acceso.

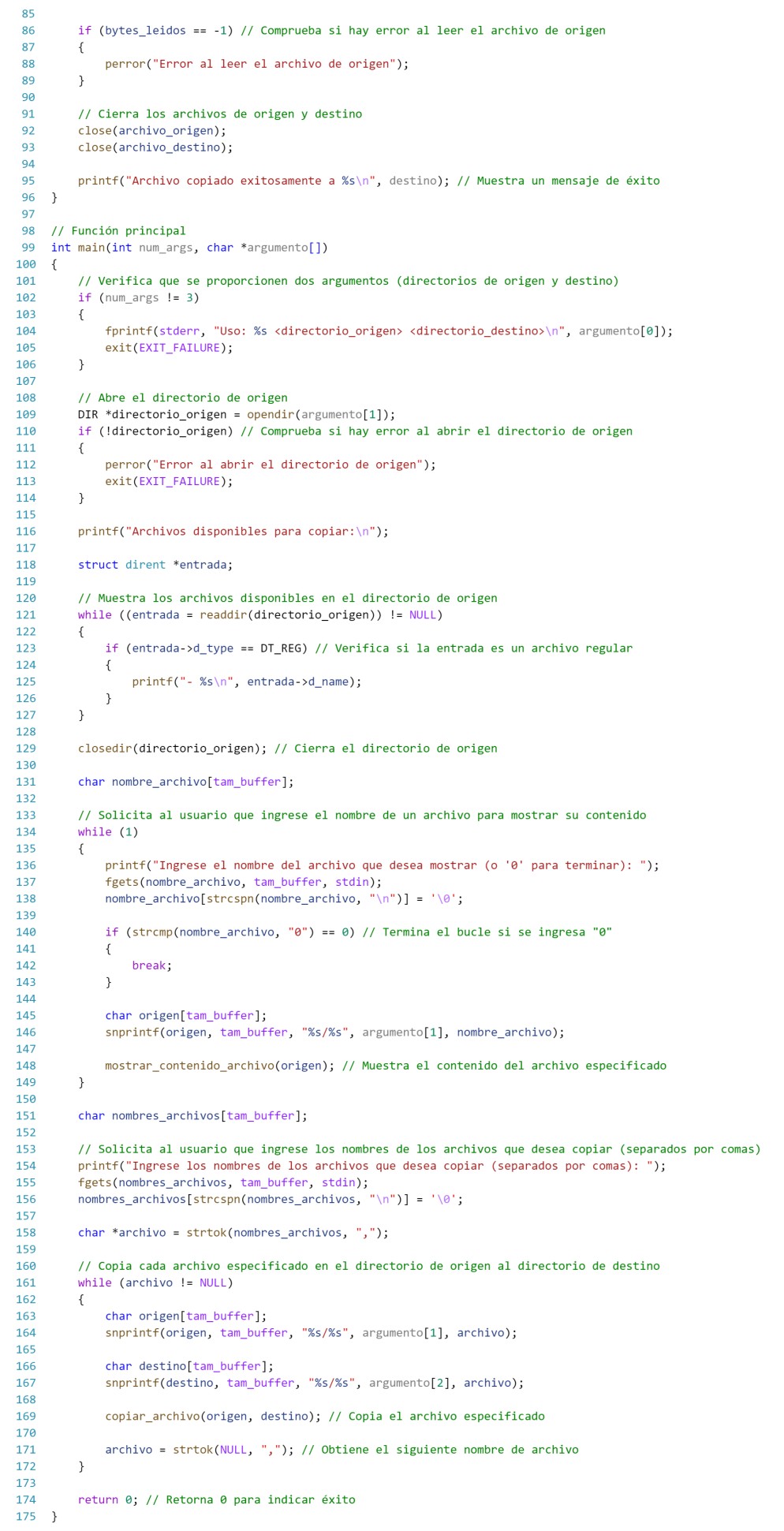




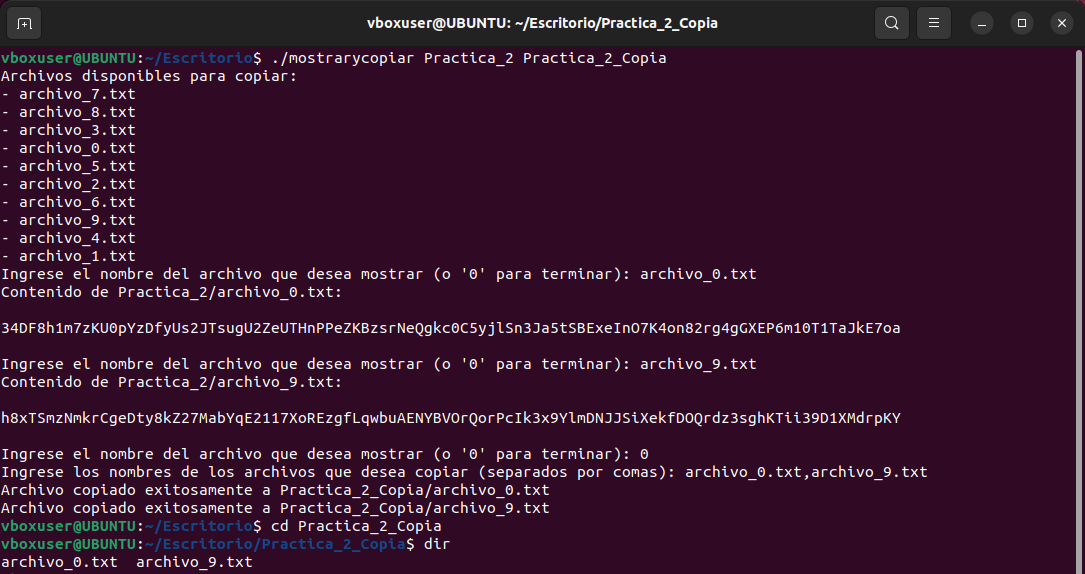


1. Una vez creados los archivos con sus contenidos por el programa del punto 8 y utilizando las llamadas al sistema revisadas para Linux que sean necesarias, desarrolle un programa en C para mostrar el contenido de un archivo seleccionado por el usuario, y que copie uno o más de los archivos creados a un directorio previamente establecido.







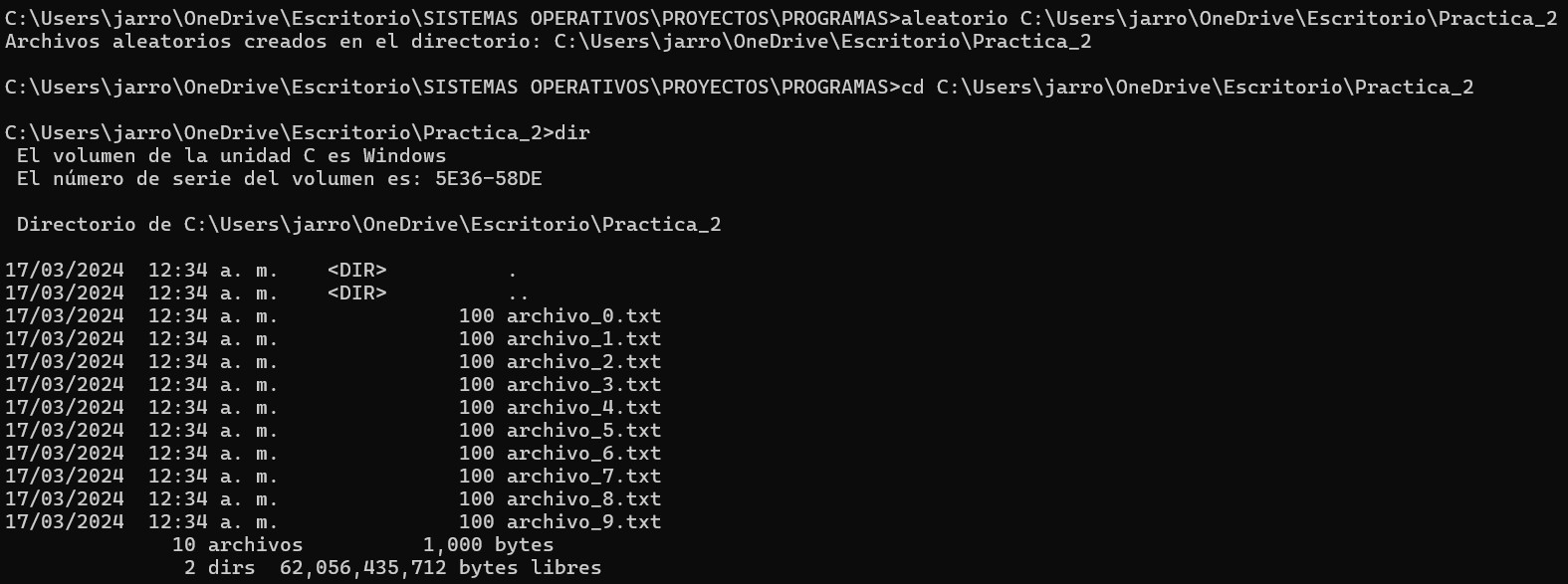


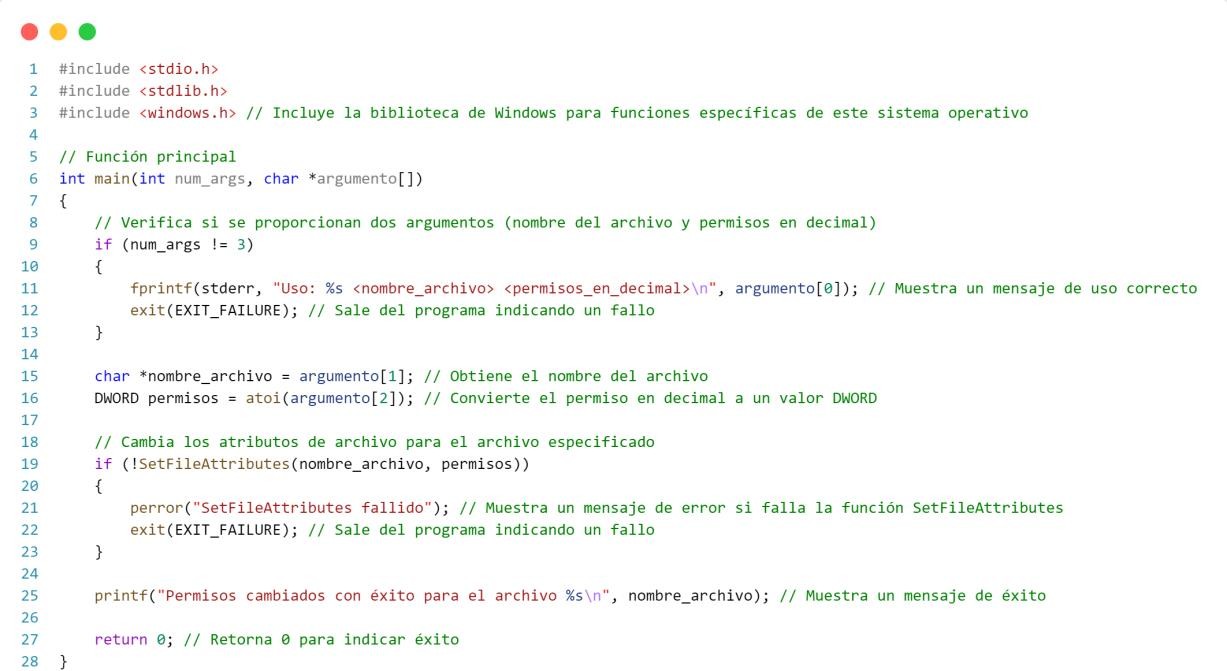
1. Desarrolle las versiones para Windows de los programas del punto 8 al 11, utilizando las llamadas al sistema para Windows que sean necesarias revisadas en el punto 7.

Archivos aleatorios:







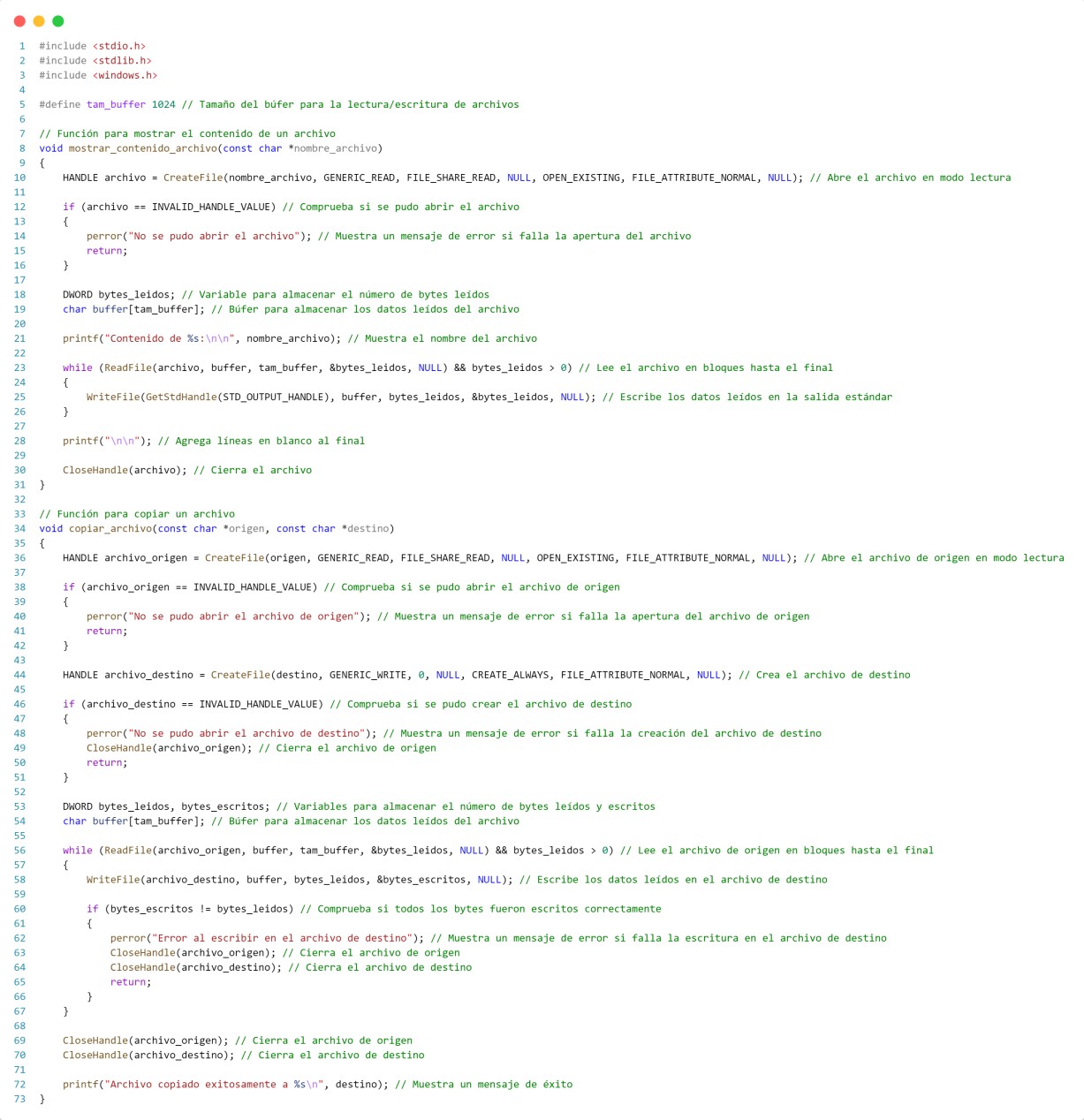
Permisos:



Información:

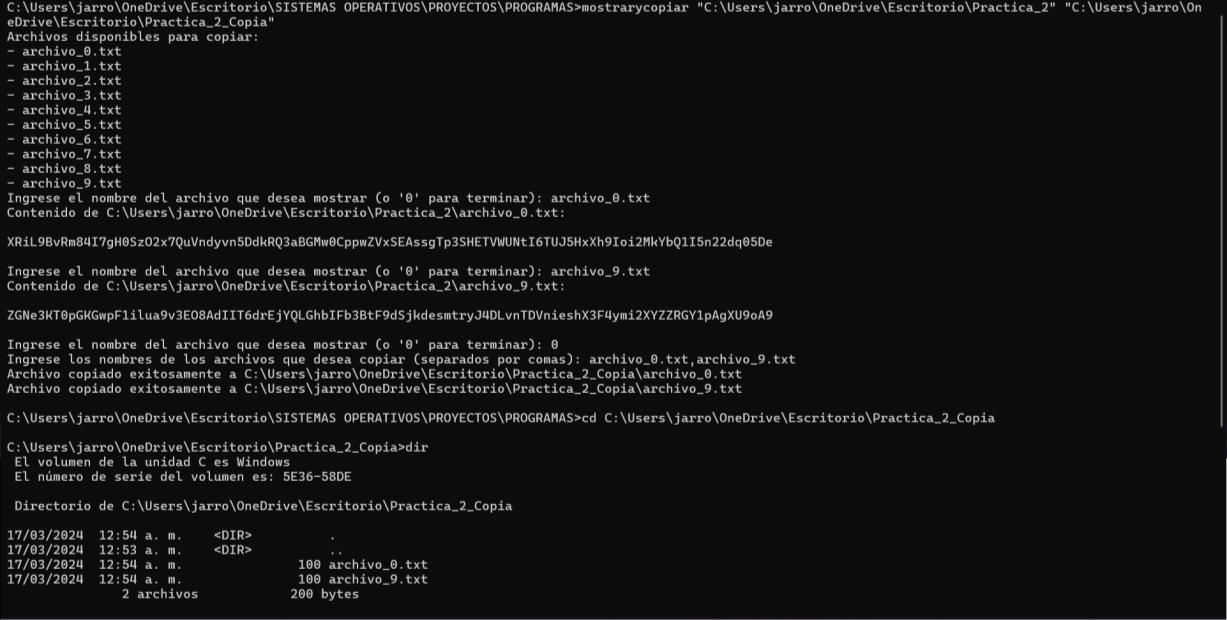


Mostrar y copiar archivos:









Conclusiones

# NOMBRE:

Durante esta práctica, he profundizado mi comprensión sobre el montaje y desmontaje de unidades USB en sistemas Linux, lo que me ha permitido familiarizarme con los procedimientos necesarios para interactuar con dispositivos externos en este entorno. Asimismo, he desarrollado habilidades en la edición de archivos utilizando gedit, lo que me ha permitido realizar modificaciones eficientemente tanto en archivos de texto como en documentos de Word. Además, he adquirido un entendimiento sólido sobre las llamadas al sistema en Linux y Windows, lo que me ha capacitado para desarrollar programas en C para diversas tareas relacionadas con el manejo de archivos.

# NOMBRE:

En esta práctica, lo que más me intereso fue el proceso de desarrollo de programas en C para interactuar con el sistema operativo y manipular archivos. Explorar las llamadas al sistema en Linux y Windows fue especialmente interesante, ya que pude comprender cómo funciona el sistema operativo a un nivel más profundo y cómo puedo aprovechar estas funcionalidades en mis propias aplicaciones. La capacidad de crear programas que generen archivos aleatorios cambie permisos, listen archivos y manipulen su contenido me pareció increíblemente útil y poderosa.



# NOMBRE:

A lo largo de esta práctica, me sumergí en el desarrollo de programas en C que interactúan con el sistema operativo y manipulan archivos, tanto en el entorno de Linux como en el de Windows. Al explorar las llamadas al sistema en ambas plataformas, descubrí fascinantes similitudes en la forma en que ambas gestionan operaciones fundamentales como la apertura, lectura y escritura de archivos. Aunque hay diferencias en la sintaxis y en la forma en que se accede a ciertas funciones, me sorprendió gratamente ver la consistencia en los conceptos subyacentes. La capacidad de crear programas que generen archivos aleatorios cambie permisos, listen archivos y manipulen su contenido en ambos sistemas operativos, resalta la flexibilidad y poder de la programación de bajo nivel.

# NOMBRE:

Durante este ejercicio práctico, he profundizado en mi comprensión del funcionamiento de unidades USB en entornos Linux y he mejorado mis habilidades para editar archivos utilizando la herramienta gedit. Además, he ampliado mis conocimientos al explorar las llamadas al sistema tanto en sistemas Linux como en Windows. Esta experiencia me ha permitido desarrollar habilidades sólidas en programación en C, lo que me ha capacitado para llevar a cabo diversas tareas, como la creación de archivos aleatorios, la modificación de permisos y la manipulación de archivos de manera efectiva.

Referencias

Google. (s.f.). Obtenido de Pdf: https://www3.uji.es/~redondo/so/Capitulo1\_IS11\_llamadas.pdf GrantMeStrength. (13 de octubre de 2021). OpenFile function (winbase.h) - Win32 apps.

Obtenido de Microsoft Learn: https://learn.microsoft.com/en- us/windows/win32/api/winbase/nf-winbase-openfile

Universidad Nacional de Rosario. (s.f.). Obtenido de https://[www.fceia.unr.edu.ar/~diegob/so/material/Llamadas\_al\_Sistema.pdf](http://www.fceia.unr.edu.ar/~diegob/so/material/Llamadas_al_Sistema.pdf)