Instituto Politécnico Nacional

ESCOM

Practica 2

**Introducción al sistema operativo Linux y Windows (2)**

**Sistemas Opreativos – 2CM17**

**Integrantes:**

Mora Ayala José Antonio

Ramírez Cotonieto Luis Fernando

Torres Carrillo Josehf Miguel Angel

Tovar Jacuinde Rodrigo

**Profesor:**

Cortes Galicia Jorge

# Introducción

A continuación, se presenta una introducción a los temas que serán profundizados dentro de esta practica mediante la implementación de código funcional relacionado a cada uno de los temas que competen este documento, así como a cada uno de los integrantes del equipo

Como todos sabemos, conforme ha pasado el tiempo las generaciones van cambiando la manera en que experimentan la evolución tecnológica que se ha dado en el mundo desde hace varios años y la cual, hoy continua, pues la tecnología cada día avanza y se van realizando nuevos descubrimientos a diario. Situación que resulta benéfica la mayoría del tiempo, pero también puede llegar a ser contraproducente, pues conforme mas acceso y facilidad se tiene a estas tecnologías y lo que conlleva interactuar con las mismas las personas en general han perdido el interés en conocer como es que funcionan o que partes son las que componen los dispositivos con los cuales interactúan día a día.  
Es por eso que comenzaremos dando un antecedente y conceptos generales y básicos para comprender de una forma más completa esta practica

## ¿Qué es un sistema operativo?

El sistema operativo es el software que coordina y dirige todos los servicios y aplicaciones que utiliza el usuario en una computadora, por eso es el más importante y fundamental. Se trata de programas que permiten y regulan los aspectos más básicos del sistema. Los sistemas operativos más utilizados son Windows, Linux, OS/2 y DOS.

Los sistemas operativos, también llamados núcleos o kernels, suelen ejecutarse de manera privilegiada respecto al resto del software, sin permitir que un programa cualquiera realice cambios de importancia sobre él que puedan comprometer su funcionamiento.

El sistema operativo es el protocolo básico de operatividad del computador, que coordina todas sus demás funciones de comunicaciones, de procesamiento, de interfaz con el usuario.

Los sistemas operativos consisten en interfaces gráficas, entornos de escritorio o gestores de ventanas que brindan al usuario una representación gráfica de los procesos en marcha. También puede ser una línea de comandos, es decir, un conjunto de instrucciones ordenado según su prioridad y que funciona en base a órdenes introducidas por el usuario.

Las primeras versiones de las computadoras no tenían sistemas operativos. En la década de los sesenta los ordenadores usaban procesamientos por lotes y fue durante estos años cuando comenzaron a desarrollarse los sistemas operativos.

## Componentes

El sistema operativo posee tres componentes esenciales o paquetes de software que permiten la interacción con el hardware:

**Sistema de archivos**. Es el registro de archivos donde adquieren una estructura arbórea.

**Interpretación de comandos**. Se logra con aquellos componentes que permiten la interpretación de los comandos, que tienen como función comunicar las órdenes dadas por el usuario en un lenguaje que el hardware pueda interpretar (sin que aquel que dé las órdenes conozca dicho lenguaje).

**Núcleo**. Permite el funcionamiento en cuestiones básicas como la comunicación, entrada y salida de datos, gestión de procesos y la memoria, entre otros.

## Para que sirve

 Gestionar **la memoria** de acceso aleatorio y ejecutar las aplicaciones, designando los recursos necesarios.  
 Administrar al[**CPU**](https://concepto.de/cpu/) gracias a un [algoritmo](https://concepto.de/algoritmo-en-informatica/) de [programación](https://concepto.de/programacion/).

 Direccionar **las entradas y salidas de datos** (a través de drivers) por medio de los [periféricos de entrada o salida](https://concepto.de/dispositivos-de-entrada-y-salida-mixtos/).  
 Administrar **la información** para el buen funcionamiento de la PC.  
 Dirigir **las autorizaciones** de uso para los usuarios.  
 Administrar **los archivos**

## Tipos de Sistema Operativo

**Según el usuario pueden ser:** multiusuario, sistema operativo que permite que varios usuarios ejecuten simultáneamente sus programas; o monousuario, sistema operativo que solamente permite ejecutar los programas de un usuario a la vez.

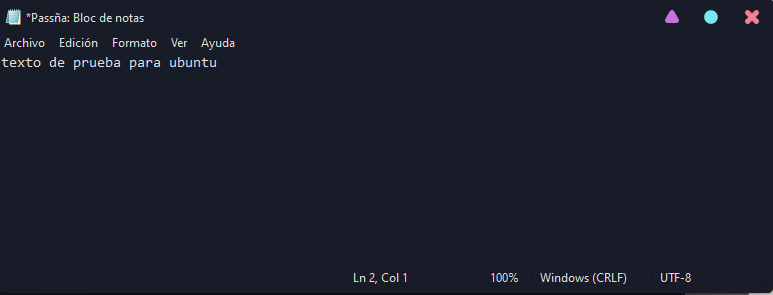
 **Según la gestión de tareas pueden ser:**monotarea, sistema operativo que solamente permite ejecutar un proceso a la vez; o multitarea, sistema operativo que puede ejecutar varios procesos al mismo tiempo.

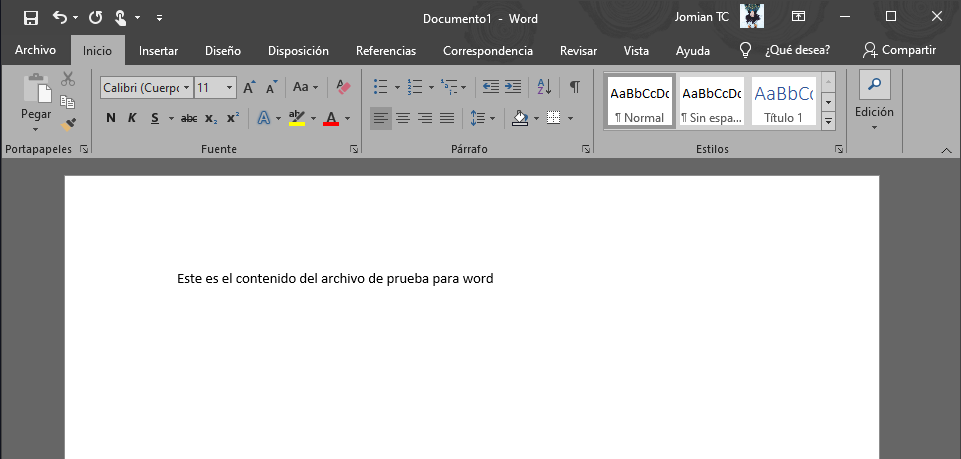
 **Según la gestión de recursos pueden ser:** centralizado, sistema operativo que solo permite utilizar los recursos de un solo ordenador; o distribuido, sistema operativo que permite ejecutar los procesos de más de un ordenador al mismo tiempo.

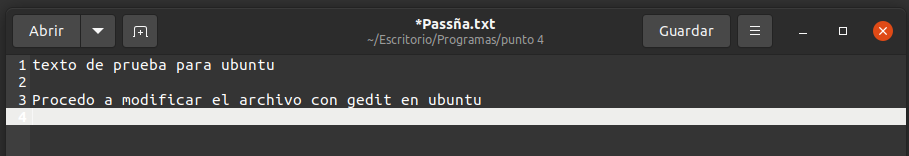
# Desarrollo Experimental

1. Cree un archivo de texto (con cualquier contenido) y un archivo en Word (con

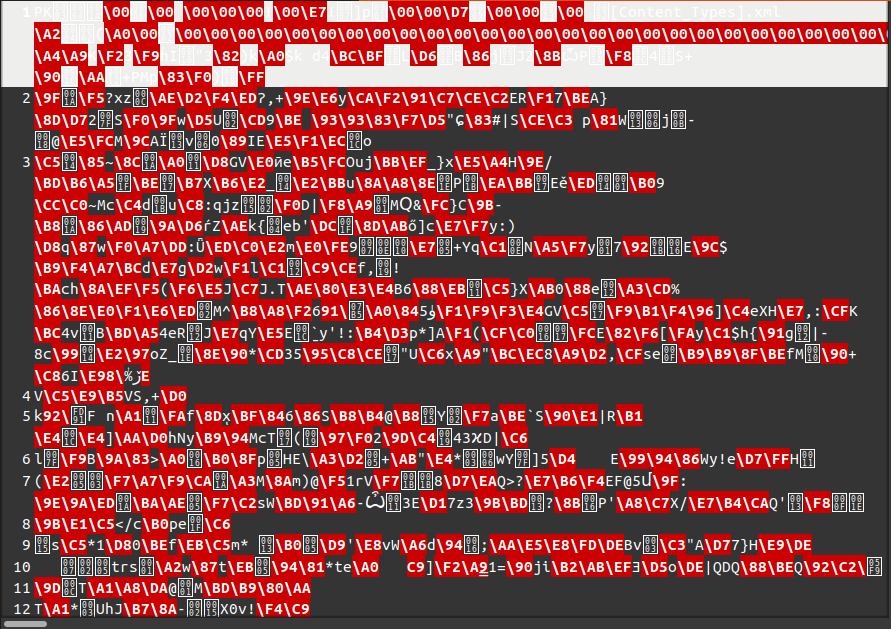
cualquier contenido) en el sistema operativo Windows y guárdelo en una memoria usb.

Archivo .txt

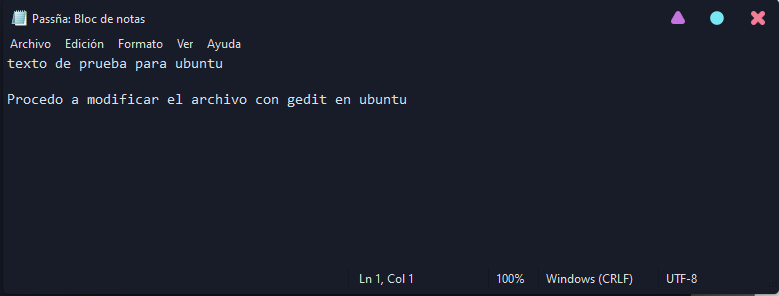
Archivo .docx

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

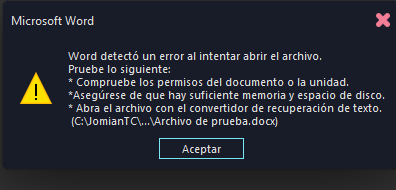
Edición de archivo .txt

Al abrir gedit tenemos el siguiente texto al abrir el .docx y procedí a quitar parte del texto para ver su modificación en Windows

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.



El archivo de texto .txt fue modificado y abierto en ambos sistemas sin problemas por lo que en este caso la prueba fue un éxito

Mientras que el archivo de .docx sufrió fallas al intentar abrirlo de nuevo en Windows mostrando el cuadro de advertencia anterior, no se pudo recuperar el archivo abriendo una versión anterior, o regresando la parte borrada el Ubuntu

Esto se debe al formato propietario de Microsoft office que usa únicamente su paqueteria y al ser un formato restringido su compilación y modificación solo puede ser hecha por el mismo software

Para el caso del txt es distinto ya que el txt es un formato de texto plano que cualquier maquina puede abrir y ejecutar sin problemas

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, opendir, y readdir.

## Llamada a Sistema LINUX

1. open

El comando open nos permite abrir un archivo en un directorio especifico, si el archivo no está creado también podemos poner como parámetro en el comando el comando creat que crear el archivo en la misma ruta

al parecer solo es un comando que podemos de manera que esté integrado en un programa ya que si intentamos poner open en nuestra terminal esta arroja que la función no existe

La función open se usa de la siguiente manera

open(const char \*pathname, int flags, mode\_t mode)

Donde pathname es un char que contiene la ruta del archivo, flags es la forma en la que vamos a abrir nuestro archivo ya sea lectura o escritura y el mode son los permisos que tendrá el archivo si es que este no se ha creado en nuestra ruta de pathname

1. close

La función close es más sencilla nos permite cerrar un archivo que se encuentre en nuestro código, de igual manera es inutilizable en la terminal solo mediante algún programa o ejecutable

La función close se usa como

close(int fd)

donde el entero fd es el archivo que tengamos abierto en ese momento

1. read

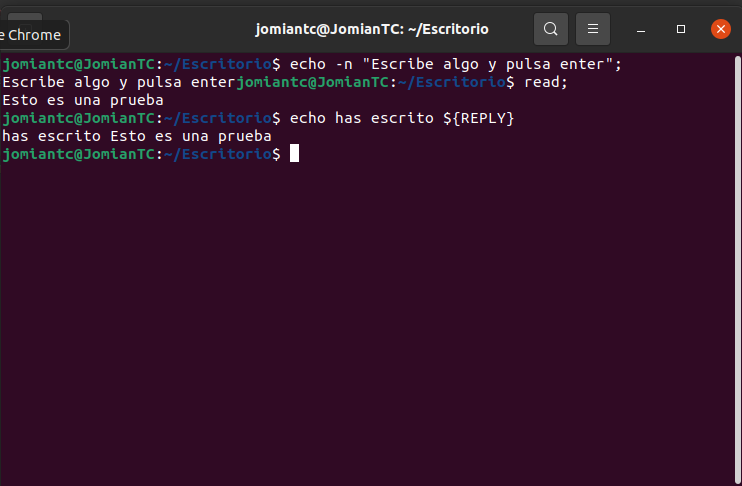
Esta función permite leer lo que hay dentro de un archivo mediante la cuenta de bytes del archivo que intentemos leer alojado en el buffer

una vez que la función retorne 0 quiere decir que ha llegado al final del archivo y podemos imprimirlo para ver lo que contiene el archivo

Podemos declarar read de la siga manera

read(int fd, void \*buf, size\_y count)

Donde fd es el archivo que vamos a leer, el buf es una estructura que esta alojada en nuestra biblioteca, y count es la variable que almacenara el número de bytes encontrados

También lo podemos usar de la siguiente manera en un terminal

1. write

Con esta función podemos escribir una cadena dentro de un archivo de texto

la forma de declarar write es la siguiente

write(int fd, const void \*buf, size\_t count);

donde fd es nuestro archivo, \*buf es la cadena que nosotros queremos ingresar en el archivo y count es el número total de bytes de la cadena

con la consola de comandos podemos mandar un mensaje a otro archivo o desplegar 2 funciones de ejemplo

1. creat

Con la función create podemos crear archivos en una ubicación que nosotros decidamos

La función create se escribe de la siguiente manera

create(const char \*pathname, mode\_t mode)

Donde pathname es la ruta con el nombre del archivo a crear y mode son los permisos que tendrá dicho archivo para ser abierto

No se puede usar el comando en una terminal

1. lseek

Esta funcion nos permite reposicionar el puntero de lectura en nuestro archivo para empezar a leer o escribir desde ahi

podemos escribir de esta manera lseek

lseek(int fd, off\_t offset, int whence);

Donde fd es el archivo, offset es a cuantos bytes queremos que se haga el recorrido y whence es desde donde partirá el offset

1. access

Con la función access podemos observar la accesibilidad de un archivo o directorio ya sea lectura, escritura o ejecutable en los casos de archivos

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

Pathname es el directorio del archivo y mode es el modo ya sea lectura o escritura o ejecutable

1. stat

Con stat podemos observar la información de un archivo como su nombre, tamaño, permisos etc.

int stat(const char \*pathname, struct stat \*statbuf);

Donde pathname es la ruta de acceso, y statbuf es una estructura donde podremos utilizar las diversas funciones de stat

1. chmod

Permite cambiar los permisos de un archivo o directorio mediante números o letras además de poner hacerlo a distintos usuarios

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

pathname ruta y mode el modo al que queremos cambiarlo

1. chown

El comando chown permite cambiar el propietario de un archivo o directorio en sistemas tipo UNIX. Puede especificarse tanto el nombre de un usuario, así como el identificador de usuario (UID) y el identificador de grupo (GID).

1. fcntl

Es una función multipropósito dependiendo de los parámetros que le demos para modificar nuestro archivo de manera eficiente

int fcntl(int fd, int cmd);

tenemos la función fcntl donde fd es el archivo y cmd es la operación que realizara el comando dependiendo del mismo

1. opendir

Permite abrir un directorio dependiendo de la ruta que nosotros mismo asignemos dentro del parámetro de la función

DIR \*opendir(const char \*name);

DIR \*fdopendir(int fd);

Podemos usarla tanto si hemos abierto nuestro archivo como si no lo hemos hecho anteriormente

1. readdir

Lee un directorio devolviendo un puntero a una estructura siguiente entrada de directorio en el flujo de directorio

struct dirent \*readdir(DIR \*dirp);

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. A través del sitio OpenGroup, investigue para que sirven las siguientes llamadas al sistema de Windows: stat, opendir, y readdir. Investigue y reporte si existe una llamada idéntica a chmod en Windows, en caso de no existir indique el motivo por el cual no existe.

## Llamada a Sistema windows

1. **OpenFile**

Crea, abre, elimina o vuelve a abrir un archivo

Los parámetros que recibe son: “nombre del archivo”, “tamaño del Buffer”, y “acción a tomar (a elegir de las listadas a continuación)”

|  |  |
| --- | --- |
| Value | Meaning |
| OF\_CANCEL 0x00000800 | Ignored. To produce a dialog box containing a **Cancel** button, use **OF\_PROMPT**. |
| OF\_CREATE 0x00001000 | Creates a new file. If the file exists, it is truncated to zero (0) length. |
| OF\_DELETE 0x00000200 | Deletes a file. |
| OF\_EXIST 0x00004000 | Opens a file and then closes it. Use this to test for the existence of a file. |
| OF\_PARSE 0x00000100 | Fills the [OFSTRUCT](https://docs.microsoft.com/en-us/windows/desktop/api/winbase/ns-winbase-ofstruct) structure, but does not do anything else. |
| OF\_PROMPT 0x00002000 | Displays a dialog box if a requested file does not exist. A dialog box informs a user that the system cannot find a file, and it contains **Retry** and **Cancel** buttons. The **Cancel** button directs **OpenFile** to return a file-not-found error message. |
| OF\_READ 0x00000000 | Opens a file for reading only. |
| OF\_READWRITE 0x00000002 | Opens a file with read/write permissions. |
| OF\_REOPEN 0x00008000 | Opens a file by using information in the reopen buffer. |
| OF\_SHARE\_COMPAT 0x00000000 | For MS-DOS–based file systems, opens a file with compatibility mode, allows any process on a specified computer to open the file any number of times.  Other efforts to open a file with other sharing modes fail. This flag is mapped to the **FILE\_SHARE\_READ**|**FILE\_SHARE\_WRITE** flags of the [CreateFile](https://docs.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-createfilea) function. |
| OF\_SHARE\_DENY\_NONE 0x00000040 | Opens a file without denying read or write access to other processes.  On MS-DOS-based file systems, if the file has been opened in compatibility mode by any other process, the function fails.  This flag is mapped to the **FILE\_SHARE\_READ**|**FILE\_SHARE\_WRITE** flags of the [CreateFile](https://docs.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-createfilea) function. |
| OF\_SHARE\_DENY\_READ 0x00000030 | Opens a file and denies read access to other processes.  On MS-DOS-based file systems, if the file has been opened in compatibility mode, or for read access by any other process, the function fails.  This flag is mapped to the **FILE\_SHARE\_WRITE** flag of the [CreateFile](https://docs.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-createfilea) function. |
| OF\_SHARE\_DENY\_WRITE 0x00000020 | Opens a file and denies write access to other processes.  On MS-DOS-based file systems, if a file has been opened in compatibility mode, or for write access by any other process, the function fails.  This flag is mapped to the **FILE\_SHARE\_READ** flag of the [CreateFile](https://docs.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-createfilea) function. |
| OF\_SHARE\_EXCLUSIVE 0x00000010 | Opens a file with exclusive mode, and denies both read/write access to other processes. If a file has been opened in any other mode for read/write access, even by the current process, the function fails. |
| OF\_VERIFY | Verifies that the date and time of a file are the same as when it was opened previously.  This is useful as an extra check for read-only files. |
| OF\_WRITE 0x00000001 | Opens a file for write access only. |

1. **CloseFile**

Como tal no encontramos la función denominada como CloseFile, si no que en Windows manejamos CloseHandle, el cual cierra un identificador de objeto abierto

BOOL CloseHandle(

HANDLE hObject

);

* Access token
* Communications device
* Console input
* Console screen buffer
* Event
* File
* File mapping
* I/O completion port
* Job
* Mailslot
* Memory resource notification
* Mutex
* Named pipe
* Pipe
* Process
* Semaphore
* Thread
* Transaction
* Waitable timer

1. **ReadFile**

Lee datos de un archivo de entrada especifico o bien de un dispositivo de entrada/salida. La lectura sucede en la posición especificada por el puntero del archivo.  
Esta función está diseñada para síncrona y asíncronas operaciones.

BOOL ReadFile(

HANDLE hFile,

LPVOID lpBuffer,

DWORD nNumberOfBytesToRead,

LPDWORD lpNumberOfBytesRead,

LPOVERLAPPED lpOverlapped

);

1. **WriteFile**

Escribe datos en un archivo p dispositivo de entrada/salida especifico

BOOL WriteFile(

HANDLE hFile,

LPCVOID lpBuffer,

DWORD nNumberOfBytesToWrite,

LPDWORD lpNumberOfBytesWritten,

LPOVERLAPPED lpOverlapped

);

1. **CreateFile**

Crea o abre un archivo o dispositivo de E/S. Los dispositivos de E/S más utilizados son los siguientes: archivo, secuencia de archivos, directorio, disco físico, volumen, búfer de consola, unidad de cinta, recurso de comunicaciones, mailslot y canalización. La función devuelve un identificador que se puede utilizar para acceder al archivo o dispositivo para varios tipos de E/S en función del archivo o dispositivo y los indicadores y atributos especificados.

Para realizar esta operación como una operación transaccionada, que da como resultado un identificador que se puede usar para la E/S transaccionada, utilice la función CreateFileTransacted.

HANDLE CreateFileA(

LPCSTR lpFileName,

DWORD dwDesiredAccess,

DWORD dwShareMode,

LPSECURITY\_ATTRIBUTES lpSecurityAttributes,

DWORD dwCreationDisposition,

DWORD dwFlagsAndAttributes,

HANDLE hTemplateFile

);

1. **SetFilePointer**

Mueve el puntero de archivo del archivo especificado.

Esta función almacena el puntero de archivo en dos valores LONG. Para trabajar con punteros de archivo que son mayores que un único valor LONG, es más fácil usar la función SetFilePointerEx.

DWORD SetFilePointer(

HANDLE hFile,

LONG lDistanceToMove,

PLONG lpDistanceToMoveHigh,

DWORD dwMoveMethod

);

1. **Stat**

La función \_stat obtiene información sobre el archivo o directorio especificado por la ruta de acceso y lo almacena en la estructura señalada por el búfer. \_stat controla automáticamente los argumentos de cadena de caracteres multibyte según corresponda, reconociendo secuencias de caracteres multibyte según la página de códigos multibyte actualmente en uso.

1. **opendir**

Nuevamente nos encontramos con la misma situación, como tal no existe una llamada al sistema denominada como opendir para este sistema operativo, por lo que optamos por usar dirent, el cual si las contiene aparte de que es compatible con UNIX también

1. **readdir**

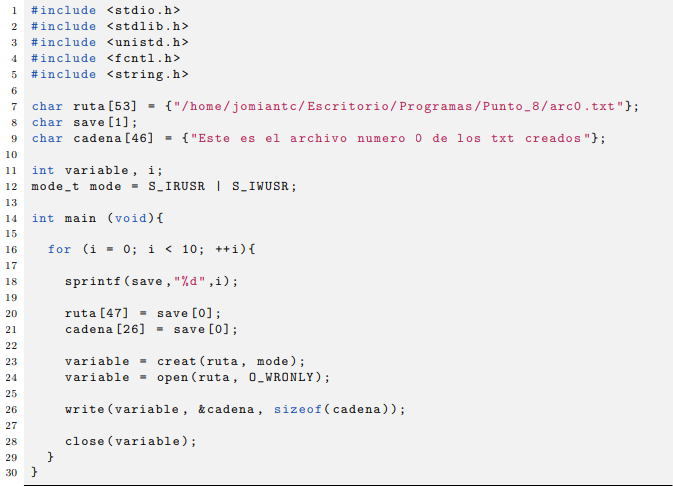
Nuevamente nos encontramos con la misma situación, como tal no existe una llamada al sistema denominada como opendir para este sistema operativo, por lo que optamos por usar dirent, el cual si las contiene aparte de que es compatible con UNIX también

## ¿chmod en windows?

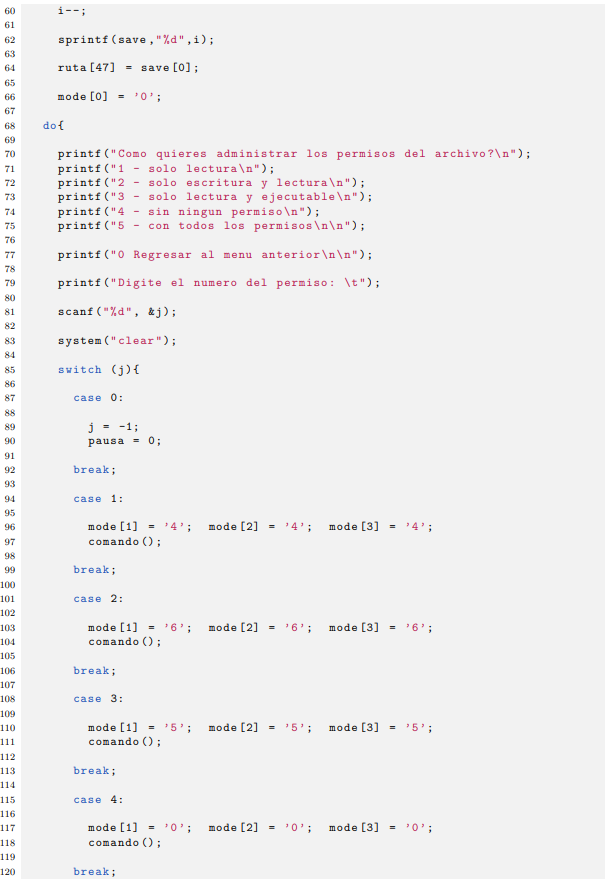
No hay nada llamado chmod en Windows porque el modelo de seguridad de Windows es diferente al de Linux. Puede usar el attribcomando para cambiar las propiedades de los objetos. (Pero son más hacia propiedades globales).

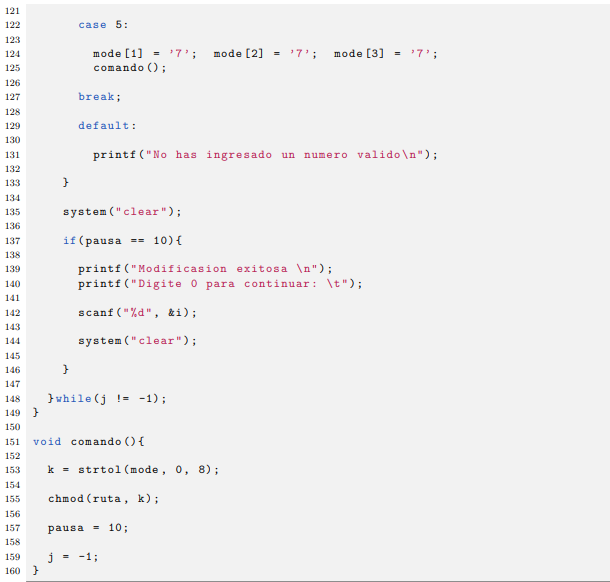
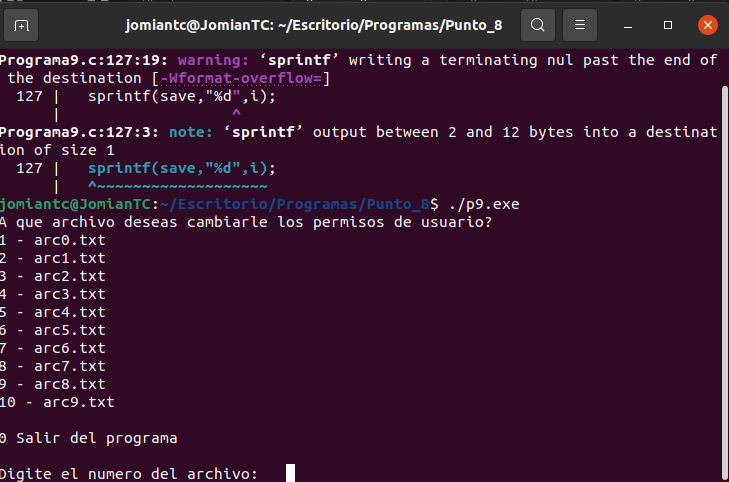
El attribcomando es la coincidencia más cercana para cosas muy básicas (solo lectura, marcas de archivo). Luego está el comando ACL (lista de control de acceso) cacls.

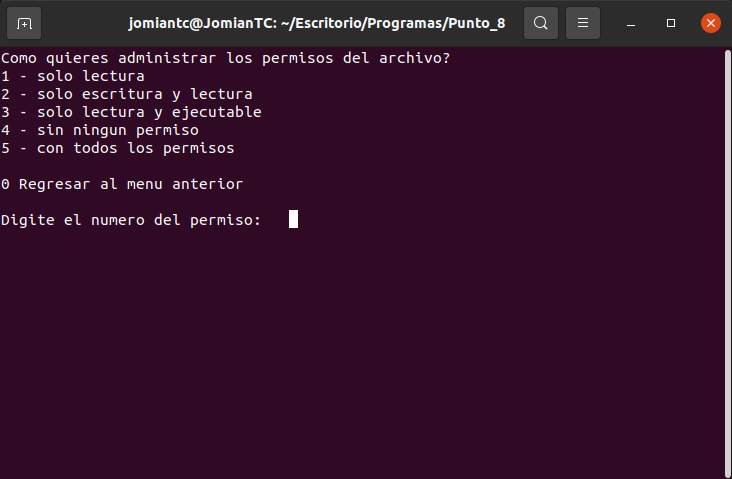
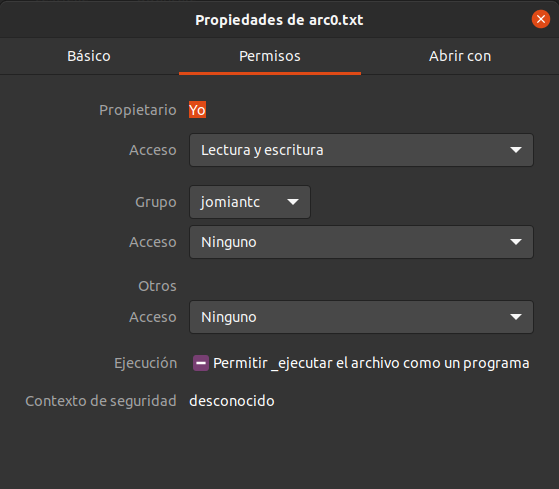
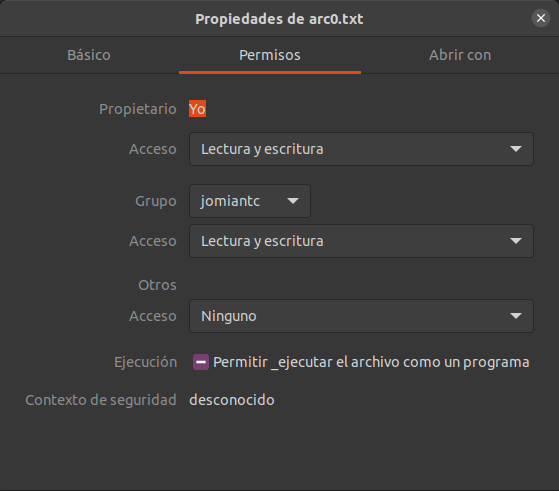
1. Utilizando únicamente las llamadas al sistema revisadas para Linux que sean necesarias, desarrolle un programa en C que cree una serie aleatoria de archivos (en una ruta especificada a través de la línea de comando), el contenido de los archivos serán cadenas que estén almacenadas en un arreglo. Restricción: Únicamente utilizar las llamadas al sistema para manipulación de archivos revisadas en el punto 6 de esta práctica.

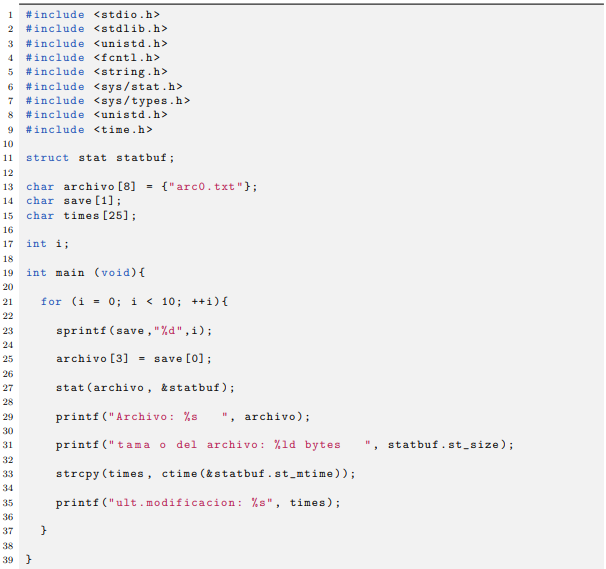
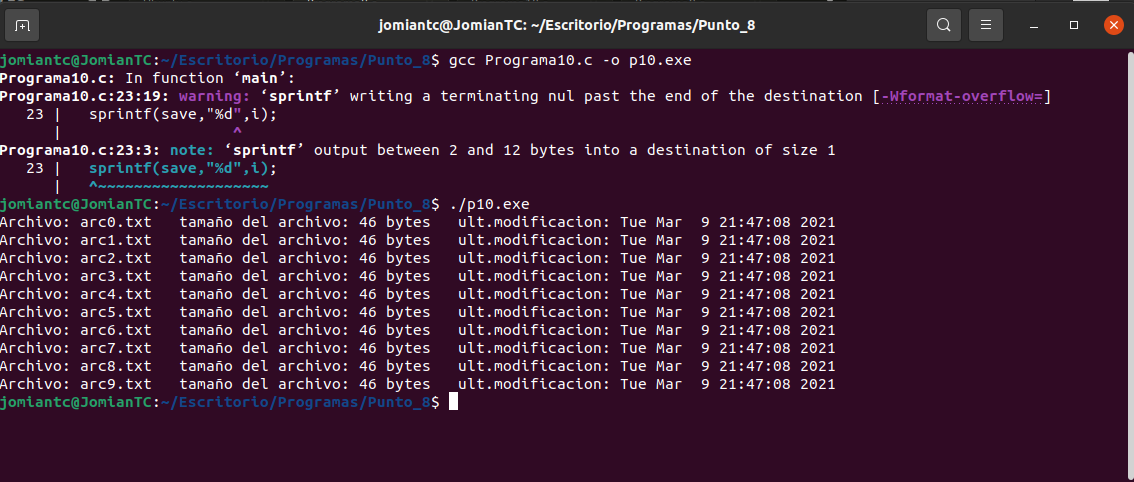


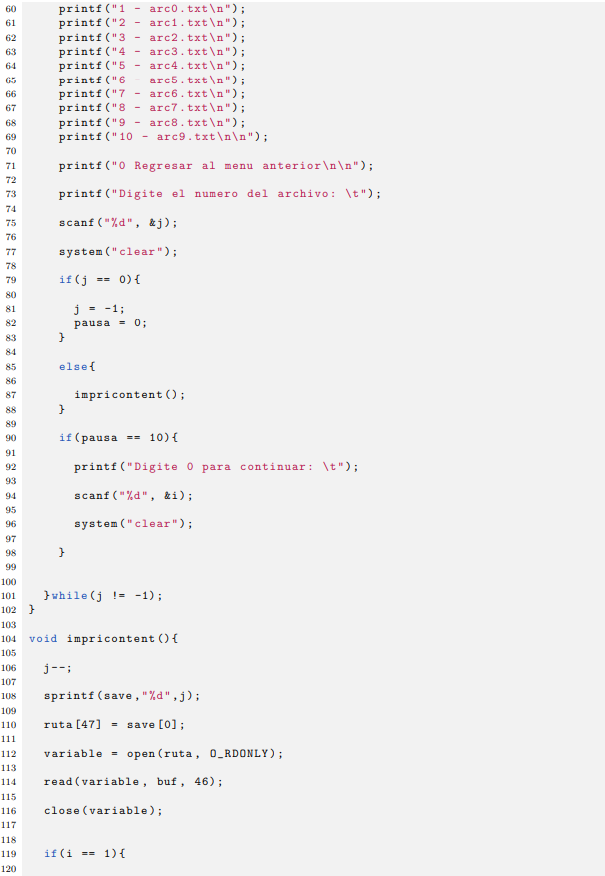
1. Una vez creados los archivos con sus contenidos por el programa del punto 8 y utilizando las llamadas al sistema revisadas para Linux que sean necesarias, desarrolle un programa en C para cambiar los permisos de un archivo seleccionado por el usuario.

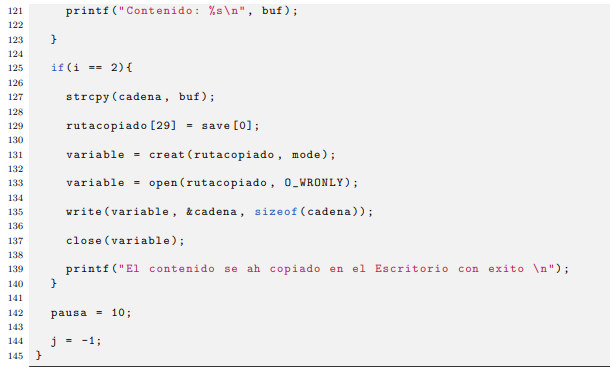
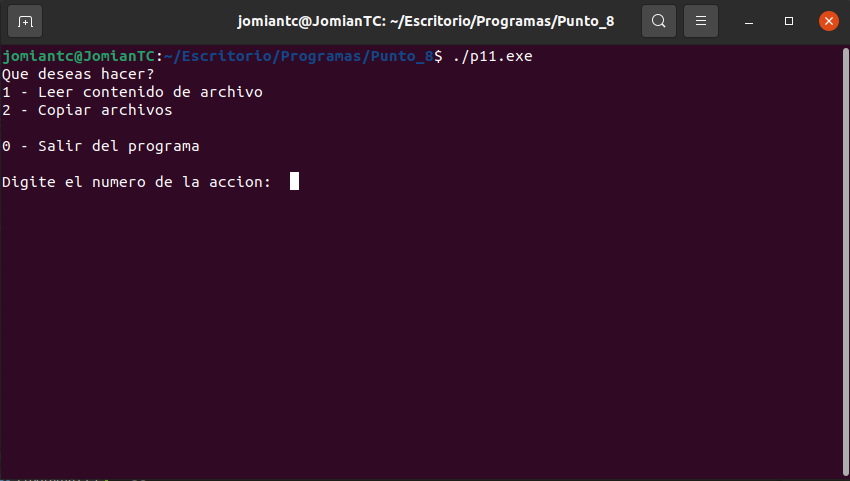
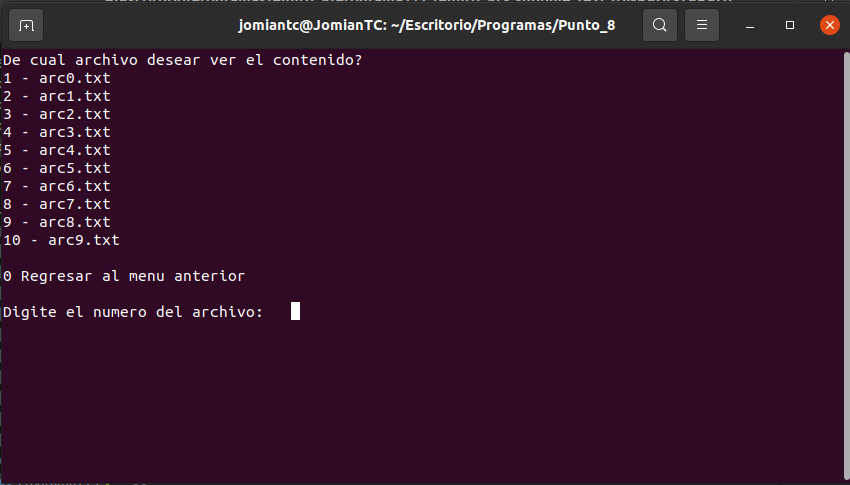


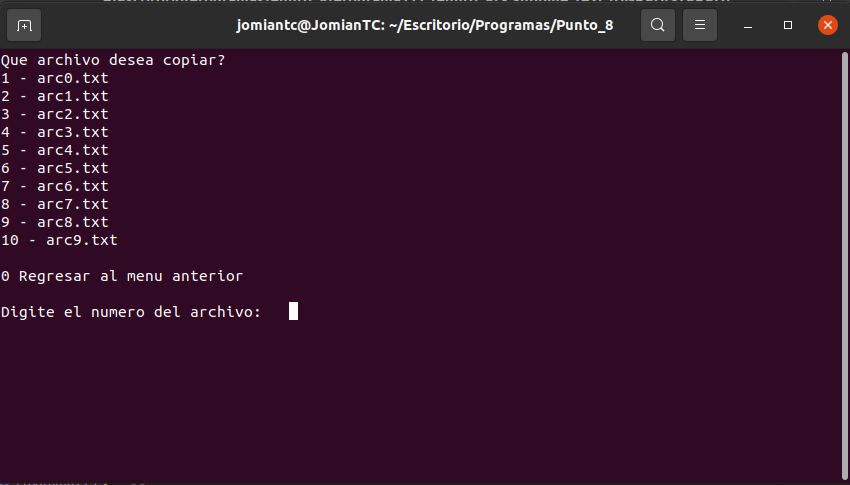




1. Una vez creados los archivos con sus contenidos por el programa del punto 8 y utilizando únicamente las llamadas al sistema revisadas para Linux que sean necesarias, desarrolle un programa en C que liste los archivos creados, mostrando su tamaño, fecha y hora de acceso.
2. Una vez creados los archivos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

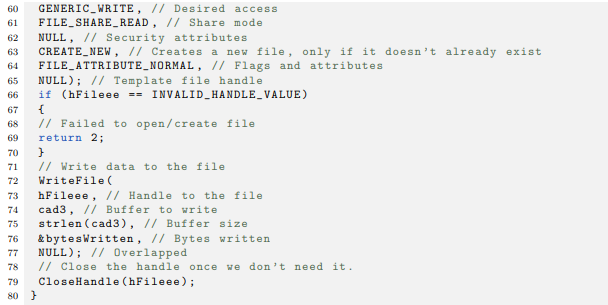
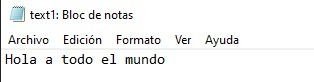


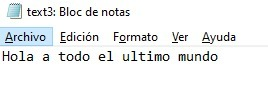


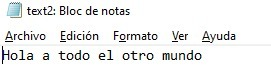


1. Desarrolle las versiones para Windows de los programas descritos en los puntos 8, 10 y utilizando únicamente las llamadas al sistema revisadas para Windows que sean necesarias.

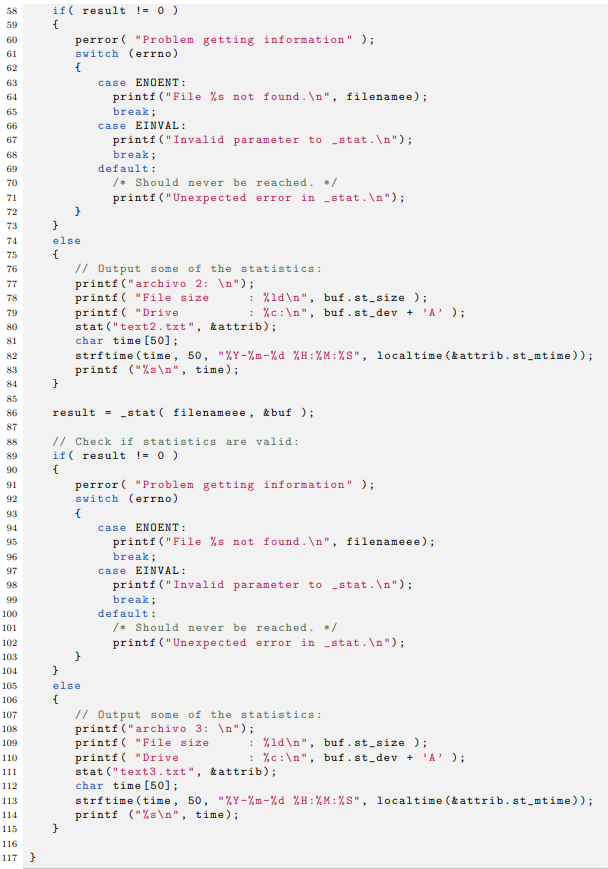
Programa 1

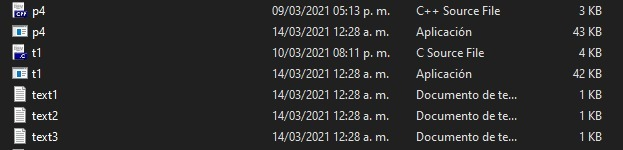
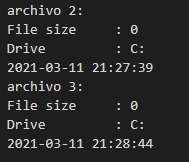
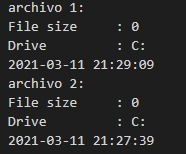
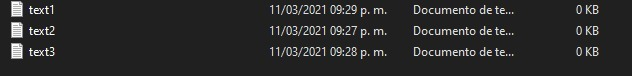




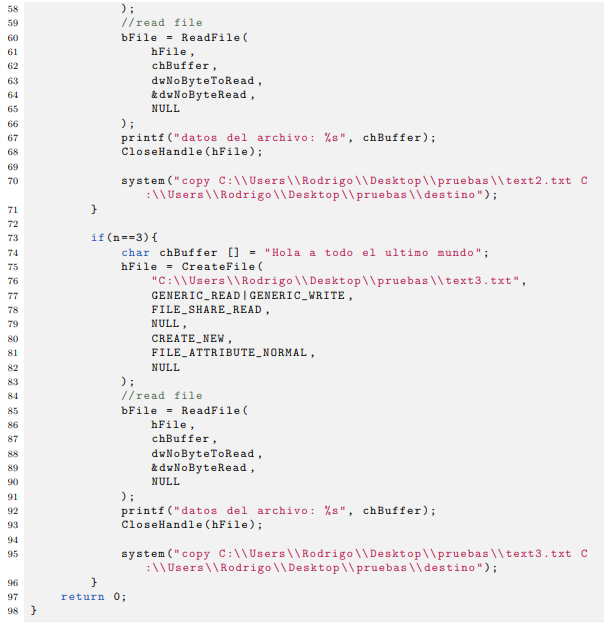
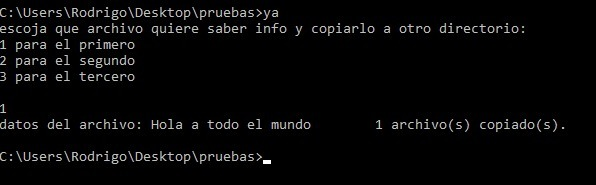


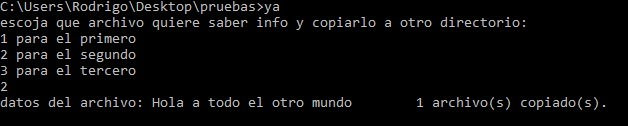
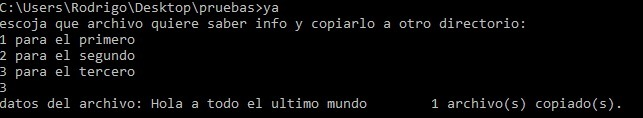
Programa 2





 Programa 3





# Conclusiones

Me parece que la practica asignada presento un buen reto para el equipo y a nivel personal, pues me hizo volver a temas que había dejado un poco abandonados por cuestiones académicas, de igual forma me permitió meterme mucho más en el ámbito de la lectura, pues uno va conociendo mucho mas acerca de los conceptos necesarios mediante libros y pdf’s, tanto los proporcionados por el profesor como aquellos a los que el equipo tuvo que recurrir de internet para poder conocer mucho mas acerca del manejo de archivos tanto de Linux como de Windows.

Me parece muy bien la forma en que realizamos la implementación de código para ambos sistemas operativos, pues nos permite relacionarnos mas con ambos entornos y poder implementar cada uno de los comandos que estos poseen (identifique de una mejor manera algunas de las diferencias, así como similitudes en ciertos comandos que tuvieron que ser usados)

***-Mora Ayala José Antonio***

En esta segunda práctica nos pudimos adentrar a visualizar de una manera más formal el proceso que lleva el funcionamiento de una computadora, apegado al del sistema operativo. Fue interesante poder conocer y leer sobre los procesos paso a paso, pues no es algo de lo que habría ya razonado y me intrigó. Hablar de su estructura interna, de su kernel, de su arquitectura y poderlos vincular a los programas tanto para Linux como para Windows, despertó en mi aun más el querer estar en Linux e investigar más de él, pues ofrece muchas más herramientas al desarrollador y simplifica de enorme manera los proyectos desde mi perspectiva.

Fue una práctica muy interesante y llena de conocimiento.

***-Ramírez Cotonieto Luis Fernando***

Esta practica me ayudo mucho a entender lo más fácil e interactivo que es Linux o en este caso Ubuntu hacia los desarrolladores en general, los programas me resultaron sencillos en interesante aun que a diferencia de Windows que tuvimos muchos problemas a la hora de la implementación de las funciones y muchos errores de compatibilidades con los IDE, Linux es un entorno que cada vez me gusta más y más, su fluidez y sencillez me agradan y al ver lo fácil que es crear código en este plataforma en el lenguaje C me hace cada ves mas pensar si puedo mudar de sistema operativo definitivamente, por otro lado la parte de la teoría también fue interesante ver como algunos comandos funcionan en la terminal muestras que otros simplemente funcionan como métodos para programas, es algo que no me imaginaba que existiera, aun que los comandos que encontramos son para cuestiones básicas de escritura, lectura, directorios y copiado de archivos también hay comandos con funciones mas especificas que nos facilitan en la obtención de información y/o manipulación de los archivos

***-Torres Carrillo Josehf Miguel Angel***

Después de realizar esta práctica me di cuenta de la diversidad de funciones que hay entre sistemas operativos, en este caso Linux y Windows, también pude notar como es más fácil el manejo de ficheros en Linux puesto que es un sistema operativo libre, en Windows me tope con problemas como que no es de muy fácil uso estas funciones de llamadas al sistema y que no todas están actualizadas porque hasta en la misma página de Microsoft no estaba actualizada su información, pero a final de cuentas también pude lograr darme cuenta de que estas mismas funciones pueden ser muy útiles para el manejo de ficheros y de directorios

***-Rodrigo Tovar Jacuinde***