

**INSTITUTO POLITÉCNICO NACIONAL**

**ESCUELA SUPERIOR DE CÓMPUTO**

**SISTEMAS OPERATIVOS**

**2CM9**

**PROFESOR: CORTES GALICIA JORGE**

**PRÁCTICA 2: INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS LINUX Y WINDOWS (2)**

**DE LOS SANTOS DÍAZ LUIS ALEJANDRO 2017630451**

**QUINTANA RUÍZ AJITZI RICARDO 2017631261**

**VÁZQUEZ MORENO MARCOS OSWALDO 2016601777**

**17 DE SEPTIEMBRE DE 2018**

**PRÁCTICA 1: INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS LINUX Y WINDOWS (2)**

**Competencias**

El alumno aprende a familiarizarse con los sistemas operativos Linux y Windows mediante el uso de la interfaz de llamadas al sistema respectiva de cada sistema operativo, a través del desarrollo de programas bajo el lenguaje C para la invocación de llamadas al sistema propias de los sistemas operativos revisados.

**DESARROLLO**

**1.- Creación de un archivo de texto y un archivo en Word**

**2.- Inicie sesión en Linux**

**3.- Verificación de la memoria usb montada en el sistema**

**4.- Edición del contenido en Word mediante el gedit**

**5.- Modificaciones realizadas en Linux**

**6.- Comandos**

**Linux**

OPEN: 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 fallará.

CLOSE: 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.

READ: es una llamada al sistema 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.

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

* *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.

CEATE: 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().

ISEEK: lseek() es una llamada al sistema que cambia la posici´on del puntero de lectura/escritura de un descriptor de archivo. Dicha posici´on 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 posición 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.

ACCESS: access() es una llamada al sistema que nos permite saber si el proceso actual puede acceder a un archivo espec´ıfico de un modo espec´ıfico.

* *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:

1. F OK: Revisa la existencia del archivo.
2. R OK: Revisa si tenemos acceso de lectura al archivo.
3. W OK: Revisa si tenemos acceso de escritura al archivo.
4. 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 falló.

STAT: stat() es una llamada al sistema que devuelve información de un archivo dada su ruta de acceso. Es importante decir que stat() recibe una ruta absoluta o relativa.

* *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:

1. *st\_mode:* Permisos actuales del archivo.
2. *st\_ino:* El “inode” del archivo.
3. *st\_dev:* El dispositivo en el cual se encuentra el archivo.
4. *st\_uid*: El ID de usuario del archivo.
5. *st\_gid:* El ID de grupo del archivo.
6. *st\_atime*: La fecha más reciente en la que el archivo fue accedido.
7. *st\_ctime:* La fecha más reciente en la que se modificaron los permisos del archivo.
8. *st\_mtime:* La fecha más reciente en la que se modificó el contenido del archivo.
9. *st\_nlink*: El número de enlaces existentes hacia este archivo.
10. *st\_size:* El tamaño del archivo en bytes.

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

CHMOD: 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 modificarán los permisos.
* *int\_fd:* El descriptor de archivo al que se le modificarán 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):

1. S\_IRUSR: 400, leer por propietario.
2. S\_IWUSR: 200, escribir por propietario.
3. S\_XUSR: 100, ejecutar por propietario.
4. S\_IRGRP: 040, leer por grupo.
5. S\_IWGRP: 020, escribir por grupo.
6. S\_IXGRP: 010, ejecutar por grupo.
7. S\_IROTH: 004, leer por otros.
8. S\_IWOTH: 002, escribir por otros.
9. S\_IXOTH: 001, ejecutar por otros.

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

CHMOD: Las llamadas al sistema chown() 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.

* const char \*pathname: La ruta absoluta o relativa del archivo al que se le modificarán los permisos.

1. Int\_fd: El descriptor de archivo al que se le modificarán los permisos.
2. uid\_t owner: ID del propietario, -1 para no cambiar.
3. 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.

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

CHDIR: La llamada al sistema chdir() cambia el directorio de trabajo actual del proceso, dado por una ruta o por un descriptor de archivo.

* const char \*path: La ruta a la nueva carpeta.
* int fd: Descriptor de archivo de la nueva carpeta.
* El valor devuelto por chdir es 0 en caso de ´exito, -1 en caso de error.

MKDIR: La llamada al sistema mkdir() crea un directorio, especificando sus permisos.

* const char \*pathname: La ruta a la nueva carpeta. El árbol de directorios previo al nombre de la última carpeta ya debe de existir.
* mode t mode: Permisos de la carpeta, definidos de la misma forma que en chmod.
* El valor devuelto por mkdir es 0 en caso de éxito, -1 en caso de error.

OPENDIR: 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.

READDIR: 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 nodo 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 (symbloic link).
* DT\_REG: archive.
* 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.

**WINDOWS**

OPENFILE: 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 función devuelve una estructura HFILE (identificador) que se usará para manipular el archivo. En caso de error, devuelve HFILE ERROR.

CLOSEFILE: La llamada al sistema CloseHandle() cierra un identificador de archivo.

* *hObject:* Un identificador correcto del archivo. En caso de éxito, devuelve un valor distinto de cero. En caso de error, devuelve cero.

READFILE: 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 guardará los bytes leídos.
* Devuelve TRUE en caso de éxito, FALSE en caso contrario.

WRITEFILE: 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 escribirá 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.

CREATEFILE: La llamada al sistema CreateFile() 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 realizará 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.

SETFILEPOINTER: 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.

CREATEDIRECTORY: La llamada al sistema CreateDirectory() crea un nuevo directorio.

* *lpPathName:* La ruta del directorio a crear.
* Devuelve TRUE en caso de éxito, FALSE en caso contrario.

SETCURRENTDIRECTORY: La llamada al sistema SetCurrentDirectory() cambia el directorio actual para el proceso actual.

* *lpPathName:* La ruta al nuevo directorio actual. Puede ser una ruta absoluta o relativa.
* Devuelve TRUE en caso de éxito, FALSE en caso contrario.

STAT

int \_stat(

const char \*path,

struct \_stat \*buffer

);

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 si se pudo obtener, -1 si ocurrió un error.

OPENDIR Y READDIR: Lo más parecido en Windows es la llamada al sistema FindFirstFile(), la cual busca en una carpeta archivos o subcarpetas de acuerdo a 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 FIND DATA que contendrá la información sobre el primer elemento encontrado.
* En caso de éxito, devuelve un identificador de búsqueda que se usará para las llamadas a las funciones *FindNextFile() o FindClose().* Si la carpeta no existe, devuelve INVALID HANDLE VALUE.
* Si no se encontró nada, devuelve ERROR FILE NOT FOUND.

EQUIVALENTE DE CHMOD: No existe un equivalente exacto de chmod en Windows, pues las definiciones de los permisos que

Linux establece no son compatibles en Windows, además de que el modelo de seguridad es diferente.

El comando más parecido es *icacls.*

*ICACLS:* Muestra o modifica listas de control de acceso discrecional (DACL) en archivos especificados, y aplica DACL almacenadas a archivos en directorios especificados.

Para guardar las DACL para todos los archivos en el directorio C: \ Windows y sus subdirectorios en el archivo ACLFile, escriba:

icacls c:\windows\\* /save aclfile /t

**OBSERVACIONES**

**ANÁLISIS CRITICO**

**CONCLUSIONES**