

# PRÁCTICA 2: INTRODUCCIÓN AL SISTEMA OPERATIVO LINUX Y WINDOWS (2)

ALUMNO: BASTIDA PRADO JAIME ARMANDO

PROFESOR: CORTES GALICIA JORGE

GRUPO: 2CM9

Marzo 2018

# Índice

<b>1. Competencias</b>	<b>3</b>
<b>2. Desarrollo</b>	<b>3</b>
2.1. Punto 5 . . . . .	3
2.2. Punto 6 . . . . .	3
2.2.1. Llamadas al sistema de Linux . . . . .	3
2.2.2. Llamadas al sistema con Windows . . . . .	5
2.3. Punto 8 Linux . . . . .	7
2.3.1. Código . . . . .	8
2.4. Punto 8 Windows . . . . .	9
2.4.1. Código . . . . .	10
2.5. Punto 9 Linux . . . . .	11
2.5.1. Código . . . . .	12
2.6. Punto 10 Linux . . . . .	18
2.6.1. Código . . . . .	19
2.7. Punto 10 Windows . . . . .	19
2.7.1. Código . . . . .	20
2.8. Punto 11 Linux . . . . .	21
2.8.1. Código . . . . .	21
2.9. Punto 11 Windows . . . . .	24
2.9.1. Código . . . . .	25
<b>3. Análisis Crítico</b>	<b>28</b>
<b>4. Observaciones</b>	<b>28</b>
<b>5. Conclusión</b>	<b>28</b>

## 1. Competencias

El alumno aprende a familiarizarse con los sistemas operativos Windows y Linux 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

## 2. Desarrollo

### 2.1. Punto 5

Después de haber creado los archivos de texto y Word en el sistema operativo Windows, procedí a instalar mi memoria usb en el sistema Linux y con el editor de texto "gedit" pude editar con éxito el archivo de texto sin embargo, cuando trate de abrir el archivo Word con gedit no pude realizar la acción. Al regresar a Windows comprobé que si se habían guardado los cambios realizados en el archivo de texto.

### 2.2. Punto 6

#### 2.2.1. Llamadas al sistema de Linux

##### ■ open

Librerías:

- `#include <sys/types.h >`
- `#include <sys/stat.h >`
- `#include <fcntl.h >`

Prototipos:

- `int open(const char *pathname, int flags);`
- `int open(const char *pathname, int flags, mode_t mode);`

Función: abre y posiblemente crea un archivo

##### ■ close

Librería: `#include <unistd.h >`

Prototipo: `int close(int fd);`

Función: cierra un descriptor de archivo

##### ■ read

Librería: `#include <unistd.h >`

Prototipo: `ssize_t read(int fd, void *buf, size_t count);`

Función: lee desde un descriptor de archivo

##### ■ write

Librería: `#include <unistd.h >`

Prototipo: `ssize_t write(int fd, const void *buf, size_t count);`

Función: escribe hacia un descriptor de archivo

##### ■ creat

Librería:

- `#include <sys/types.h >`

- `#include <sys/stat.h >`
- `#include <fcntl.h >`

Prototipo: `int creat(const char *pathname, mode_t mode);`

Función: abre y posiblemente crea un archivo

#### ■ **lseek**

Librería:

- `#include <sys/types.h >`
- `#include <unistd.h >`

Prototipo: `off_t lseek(int fd, off_t offset, int whence);`

Función: reposiciona el offset de un archivo

#### ■ **access**

Librería: `#include <unistd.h >`

Prototipo: `int access(const char *pathname, int mode);`

Función: checa los permisos de usuario para un archivo

#### ■ **stat**

Librería:

- `#include <sys/types.h >`
- `#include <unistd.h >`

Prototipo: `int stat(const char *pathname, struct stat *statbuf);`

Función: regresa el estatus del archivo

#### ■ **chmod**

Librería: `#include <sys/stat.h >`

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

Función: cambia los permisos para un archivo

#### ■ **chown**

Librería: `#include <unistd.h >`

Prototipo: `int access(const char *pathname, uid_t owner, gid_t group);`

Función: cambia la propiedad de un archivo

#### ■ **fcntl**

Librerías:

- `#include <unistd.h >`
- `#include <fcntl.h >`

Prototipo: `int fcntl(int fd, int cmd, ... /* arg */);`

Función: manipula el descriptor de un archivo

#### ■ **chdir**

Librería: `#include <unistd.h >`

Prototipo: `int chdir(const char *path);`

Función: cambia el directorio con el cual trabajar

#### ■ **mkdir**

Librerías:

- `#include <sys/stat.h >`
- `#include <sys/types.h >`

Prototipo: `int mkdir(const char *pathname, mode_t mode);`

Función: crea un directorio

#### ■ **opendir**

Librerías:

- `#include <sys/types.h >`
- `#include <dirent.h >`

Prototipo: `DIR *opendir(const char *name);`

Función: abre un directorio

#### ■ **readdir**

Librería: `#include <dirent.h >`

Prototipo: `struct dirent *readdir(DIR *dirp);`

Función: lee un directorio

### 2.2.2. Llamadas al sistema con Windows

#### ■ **OpenFile**

Librería: `#include <Windows.h >`

Prototipo: `HFILE WINAPI OpenFile(  
_In_ LPCSTR lpFileName,  
_Out_ LPOFSTRUCT lpReOpenBuff,  
_In_ UINT uStyle );`

Función: crea, abre, reabre o elimina un archivo

#### ■ **CloseHandle**

Librería: `#include <Windows.h >`

Prototipo: `BOOL WINAPI CloseHandle(  
_In_ HANDLE hObject  
);`

Función: cierra un manipulador de objeto abierto

#### ■ **ReadFile**

Librería: `#include <Windows.h >`

Prototipo: `BOOL WINAPI ReadFile( _In_ HANDLE hFile,  
_Out_ LPVOID lpBuffer,  
_In_ DWORD nNumberOfBytesToRead,  
_Out_opt_ LPDWORD lpNumberOfBytesRead,  
_Inout_opt_ LPOVERLAPPED lpOverlapped );`

Función: lee información de un archivo

#### ■ **WriteFile**

Librería: `#include <Windows.h>`

Prototipo: `BOOL WINAPI WriteFile( _In_ HANDLE hFile,  
_In_ LPCVOID lpBuffer,  
_In_ DWORD nNumberOfBytesToWrite,  
_Out_opt_ LPDWORD lpNumberOfBytesWritten,  
_Inout_opt_ LPOVERLAPPED lpOverlapped );`

Función: escribe información hacia un archivo

#### ■ **CreateFile**

Librería: `#include <Windows.h>`

Prototipo: `HANDLE WINAPI CreateFile( _In_ LPCTSTR lpFileName,  
_In_ DWORD dwDesiredAccess,  
_In_ DWORD dwShareMode,  
_In_opt_ LPSECURITY_ATTRIBUTES lpSecurityAttributes,  
_In_ DWORD dwCreationDisposition,  
_In_ DWORD dwFlagsAndAttributes,  
_In_opt_ HANDLE hTemplateFile );`

Función: crea o abre un archivo

#### ■ **SetFilePointer**

Librería: `#include <Windows.h>`

Prototipo: `DWORD WINAPI SetFilePointer( _In_ HANDLE hFile,  
_In_ LONG lDistanceToMove,  
_Inout_opt_ PLONG lpDistanceToMoveHigh,  
_In_ DWORD dwMoveMethod );`

Función: mueve el puntero de un archivo especificado

#### ■ **CreateDirectory**

Librería: `#include <Windows.h>`

Prototipo: `BOOL WINAPI CreateDirectory(  
_In_ LPCTSTR lpPathName,  
_In_opt_ LPSECURITY_ATTRIBUTES lpSecurityAttributes );`

Función: crea un nuevo directorio

#### ■ **SetCurrentDirectory**

Librería: `#include <Windows.h>`

Prototipo: `BOOL WINAPI SetCurrentDirectory(  
_In_ LPCTSTR lpPathName );`

Función: cambia directorio actual

#### ■ **stat**

Librería: `#include <sys/stat.h>`

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

Función: obtiene el estatus de un archivo

#### ■ opendir

Librería: `#include <dirent.h>`

Prototipo: `DIR *opendir(const char *dirname);`

Función: abre un directorio

#### ■ readdir

Librería: `#include <dirent.h>`

Prototipo: `struct dirent *readdir(DIR *dirp);`

Función: lee un directorio

**La función en Windows más parecida a *chmod* es *icalcs*, no hay un equivalente a *chmod* en Windows, esto se debe a que Linux y Windows usan los atributos para diferentes propósitos, tienen un modelo de seguridad diferente. Cabe resaltar que esta no es una llamada al sistema**

### 2.3. Punto 8 Linux

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 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.

Al correr el programa lo hacemos indicando el nombre del archivo ejecutable y el nombre del directorio que queremos crear:

```
james@james-Lenovo-ideapad-320-15ABR:~/Documents/ESCOM_SEMESTRE_5/2CM9_SISTEMAS_
PERATIVOS/1_Parcial/2_Práctica$ ./a.out E
james@james-Lenovo-ideapad-320-15ABR:~/Documents/ESCOM_SEMESTRE_5/2CM9_SISTEMAS_
PERATIVOS/1_Parcial/2_Práctica$
```

Figura 1:

Veremos entonces los cambios reflejados dentro del directorio que nombramos:

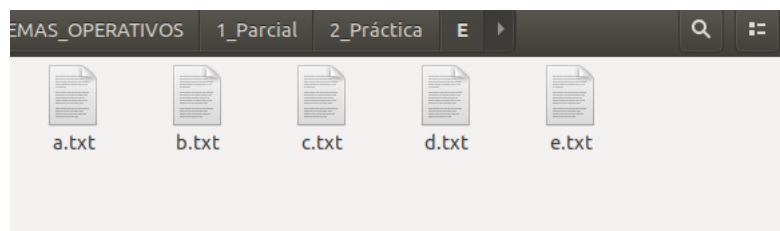


Figura 2:

### 2.3.1. Código

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

//PARA PODER CAMBIAR EL NUMERO DE CADENAS, LA LONGITUD DE CADA UNA DE ELLAS, Y EL
//NO DE ARCHIVOS
#define NO_STRINGS 10
#define LENGTH_STRINGS 100
#define NO_FILES 5

int main(int argc, char const *argv[])
{
    int file_d[NO_FILES];
    short i, j;
    char *strings[NO_STRINGS];
    char *files[NO_FILES];
    char filea[6] = "a.txt";
    char fileb[6] = "b.txt";
    char filec[6] = "c.txt";
    char filed[6] = "d.txt";
    char filee[6] = "e.txt";

    srand((unsigned) time(NULL));

    //CREANDO LAS CADENAS CON LETRAS ALEATORIAS A Y B
    for(i = 0; i < NO_STRINGS; i++)
    {
        strings[i] = malloc(LENGTH_STRINGS);
        for(j = 0; j < LENGTH_STRINGS - 1; j++)
        {
            if(rand() % 2)
                strings[i][j] = 'a';
            else
                strings[i][j] = 'b';
        }
        strings[i][j] = '\0';
    }

    //ASIGNANDO LOS NOMBRES DE LOS ARCHIVOS AL ARREGLO DE APUNTADES A
    //Cadenas para posterior proceso
    files[0] = filea;
    files[1] = fileb;
    files[2] = filec;
    files[3] = filed;
    files[4] = filee;

    //CREANDO EL DIRECTORIO CON LA RUTA ESPECIFICADA DESDE CONSOLA
    if((mkdir(argv[1], 0777)) == -1)
    {
        printf("Error: _Cannot_create_the_directory.\n");
        exit(EXIT_FAILURE);
    }

    //CAMBIANDO DE DIRECTORIO
    if((chdir(argv[1])) == -1)
    {
        printf("Error: _Cannot_change_the_directory.\n");
        exit(EXIT_FAILURE);
    }

    //CREANDO LOS ARCHIVOS DENTRO DEL DIRECTORIO Y ESCRIBIENDO EN ELLOS LAS
    //Cadenas creadas anteriormente
```



```

    for(i = 0; i < NO_FILES; i++)
    {
        if((file_d[i] = creat(files[i], S_IRWXU)) == -1)
        {
            printf("Error: _Cannot_create_the_file.\n");
            exit(EXIT_FAILURE);
        }
        write(file_d[i], strings[rand() % NO_STRINGS], LENGTH_STRINGS - 1);
        close(file_d[i]);
    }

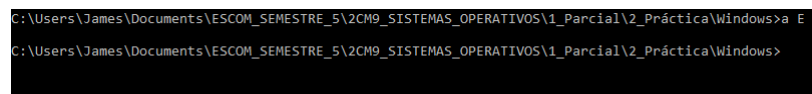
    return 0;
}

```

## 2.4. Punto 8 Windows

Utilizando las llamadas al sistema revisadas para Windows 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 directorio no debe existir previamente), el contenido de los archivos serán cadenas que estén almacenadas en un arreglo.

Al correr el programa lo hacemos indicando el nombre del archivo ejecutable y el nombre del directorio que queremos crear:



```

C:\Users\James\Documents\ESCOM_SEMESTRE_5\2CM9_SISTEMAS_OPERATIVOS\1_Parcial\2_Práctica\Windows>a E
C:\Users\James\Documents\ESCOM_SEMESTRE_5\2CM9_SISTEMAS_OPERATIVOS\1_Parcial\2_Práctica\Windows>

```

Figura 3:

Veremos entonces los cambios reflejados dentro del directorio que nombramos:

Figura 4:

### 2.4.1. Código

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <Windows.h>

//PARA PODER CAMBIAR EL NUMERO DE CADENAS, LA LONGITUD DE CADA UNA DE ELLAS, Y EL
//NO DE ARCHIVOS
#define NO_STRINGS 10
#define LENGTH_STRINGS 100
#define NO_FILES 5

int main(int argc, char const *argv[])
{
    HANDLE file_d[NO_FILES];
    short i, j;
    char *strings[NO_STRINGS];
    char *files[NO_FILES];
    char filea[6] = "a.txt";
    char fileb[6] = "b.txt";
    char filec[6] = "c.txt";
    char filed[6] = "d.txt";
    char filee[6] = "e.txt";

    srand((unsigned) time(NULL));

    //CREANDO LAS CADENAS CON LETRAS ALEATORIAS A Y B
    for(i = 0; i < NO_STRINGS; i++)
    {
        strings[i] = malloc(LENGTH_STRINGS);
        for(j = 0; j < LENGTH_STRINGS - 1; j++)
        {
            if(rand() % 2)
                strings[i][j] = 'a';
            else
                strings[i][j] = 'b';
        }
        strings[i][j] = '\0';
    }

    //ASIGNANDO LOS NOMBRES DE LOS ARCHIVOS AL ARREGLO DE APUNTADES A
    //Cadenas para posterior proceso
    files[0] = filea;
    files[1] = fileb;
    files[2] = filec;
    files[3] = filed;
    files[4] = filee;

    //CREANDO EL DIRECTORIO CON LA RUTA ESPECIFICADA DESDE CONSOLA
    if((CreateDirectory(argv[1], NULL)) == 0)
    {
        printf("Error: _Cannot_create_the_directory.\n");
        exit(EXIT_FAILURE);
    }

    //CAMBIANDO DE DIRECTORIO
    if((SetCurrentDirectory(argv[1])) == 0)
    {
        printf("Error: _Cannot_change_the_directory.\n");
        exit(EXIT_FAILURE);
    }

    //CREANDO LOS ARCHIVOS DENTRO DEL DIRECTORIO Y ESCRIBIENDO EN ELLOS LAS
    //Cadenas creadas anteriormente
    for(i = 0; i < NO_FILES; i++)
    {
        if((file_d[i] = CreateFile(files[i], GENERIC_READ | GENERIC_WRITE,
            FILE_SHARE_WRITE, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL,
```

```

        NULL)) == INVALID_HANDLE_VALUE)
    {
        printf("Error: _Cannot_create_the_file.\n");
        exit(EXIT_FAILURE);
    }
    WriteFile(file_d[i], strings[rand() % NO_STRINGS], LENGTH_STRINGS -
        1, NULL, NULL);
    CloseHandle(file_d[i]);
}

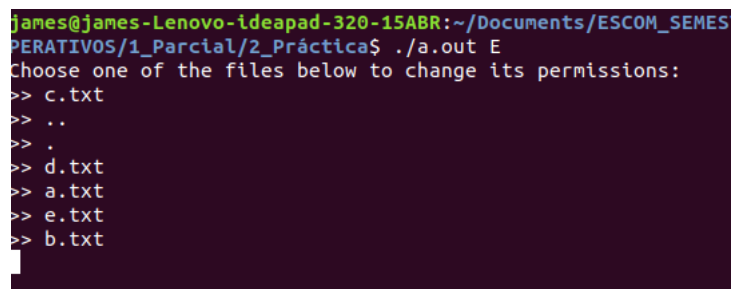
return 0;
}

```

## 2.5. Punto 9 Linux

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.

Al correr el programa lo hacemos indicando el nombre del archivo ejecutable y el nombre del directorio que queremos ver para cambiar los permisos de los archivos dentro de él, nos aparecerá un menú como el siguiente:



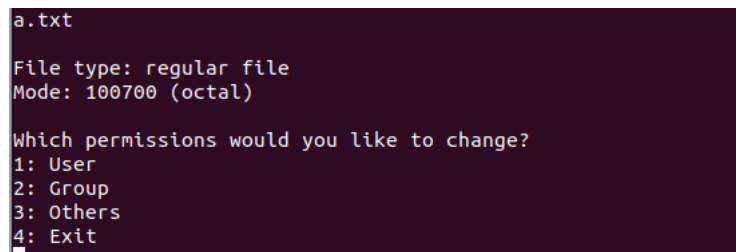
```

james@james-Lenovo-ideapad-320-15ABR:~/Documents/ESCOM_SEMESTRATIVOS/1_Parcial/2_Práctica$ ./a.out E
Choose one of the files below to change its permissions:
>> c.txt
>> ..
>> .
>> d.txt
>> a.txt
>> e.txt
>> b.txt

```

Figura 5:

Escribimos el nombre del archivo que queremos cambiar y nos aparecerá el tipo de archivo que seleccionamos y los permisos actuales que tiene además de un menú como el siguiente, preguntándonos si queremos cambiar los permisos de usuario, grupo u otros:



```

a.txt
File type: regular file
Mode: 100700 (octal)
Which permissions would you like to change?
1: User
2: Group
3: Others
4: Exit

```

Figura 6:

Nos aparecerá otro menú preguntandonos cual de los permisos de ese archivo de esa clase queremos cambiar:

```
Which user permissions would you like to change?
1: Set read, write and execute permissions
2: Set read permission
3: Set write permission
4: Set execute permission
5: Set read and write permissions
6: Set read and execute permissions
7: Set write and execute permissions
1
Done!
```

Figura 7:

### 2.5.1. Código

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

int main(int argc, char const *argv[])
{
    DIR *directory_stream;
    struct dirent *directory_entry;
    char file_name[FILENAME_MAX];
    struct stat file_status;
    short option1, option2, option_correct2 = 1;

    //ABRIMOS EL DIRECTORIO QUE INGRESO EL USUARIO AL CORRER EL PROGRAMA EN LA
    //LINEA DE COMANDOS
    if((directory_stream = opendir(argv[1])) == NULL)
    {
        printf("Error: _Cannot _open _the _directory .\n");
        exit(EXIT_FAILURE);
    }

    //RECORREMOS EL DIRECTORIO E IMPRIMIMOS EL NOMBRE DE SUS CONTENIDOS
    printf("Choose _one _of _the _files _below _to _change _its _permissions: _\n");
    while((directory_entry = readdir(directory_stream)) != NULL)
        printf(">> _%s\n", directory_entry->d_name);

    //LEEMOS EL NOMBRE DEL ARCHIVO EL USUARIO QUIERE CAMBIAR SUS PERMISOS
    scanf("%s", file_name);
    printf("\n");

    //CAMBIANDO DE DIRECTORIO PARA PODER LLAMAR STAT SIN NECESIDAD DE AGREGAR
    //EL NOMBRE DEL ARCHIVO A LA RUTA DEL DIRECTORIO (PROCESO LIOSO)
    if((chdir(argv[1])) == -1)
    {
        printf("Error: _Cannot _change _the _directory .\n");
        exit(EXIT_FAILURE);
    }

    //OBTENEMOS EL STATUS DEL ARCHIVO
    if((stat(file_name, &file_status)) == -1)
    {
        printf("Error: _Cannot _open _file _status .\n");
        exit(EXIT_FAILURE);
    }

    //IMPRIMIMOS INFORMACION ACERCA DEL ARCHIVO
```

```

printf("File_type:_");
switch(file_status.st_mode & S_IFMT)
{
    case S_IFBLK: printf("block_device\n"); break;
    case S_IFCHR: printf("character_device\n"); break;
    case S_IFDIR: printf("directory\n"); break;
    case S_IFIFO: printf("FIFO/pipe\n"); break;
    case S_IFLNK: printf("symlink\n"); break;
    case S_IFREG: printf("regular_file\n"); break;
    case S_IFSOCK: printf("socket\n"); break;
    default: printf("unknown?\n"); break;
}
printf("Mode:_%o_(octal)\n", (unsigned long) file_status.st_mode);

for (;;)
{
    printf("\nWhich_permissions_would_you_like_to_change?\n");
    printf("1:_User\n");
    printf("2:_Group\n");
    printf("3:_Others\n");
    printf("4:_Exit\n");
    scanf("%d", &option1);
    option_correct2 = 1;
    switch(option1)
    {
        case 1:
            while(option_correct2 == 1)
            {
                printf("\nWhich_user_permissions_would_you_like_to_change?\n");
                printf("1:_Set_read_write_and_execute_permissions\n");
                printf("2:_Set_read_permission\n");
                printf("3:_Set_write_permission\n");
                printf("4:_Set_execute_permission\n");
                printf("5:_Set_read_and_write_permissions\n");
                printf("6:_Set_read_and_execute_permissions\n");
                printf("7:_Set_write_and_execute_permissions\n");
                scanf("%d", &option2);
                switch(option2)
                {
                    case 1:
                        if((chmod(file_name, S_IRWXU)) == -1)
                        {
                            printf("Error:_cannot_change_user_requested_permissions\n");
                            ;
                            break;
                        }
                        option_correct2 = 0;
                    break;
                    case 2:
                        if((chmod(file_name, S_IRUSR)) == -1)
                        {
                            printf("Error:_cannot_change_user_requested_permissions\n");
                            ;
                            break;
                        }
                        option_correct2 = 0;
                    break;
                }
            }
        break;
    }
}

```

```

case 3:
    if((chmod(file_name ,
        S_IWUSR)) == -1)
    {
        printf("Error:~
            cannot_change~
            user_requested~
            permissions\n")
        ;
        break;
    }
    option_correct2 = 0;
break;
case 4:
    if((chmod(file_name ,
        S_IXUSR)) == -1)
    {
        printf("Error:~
            cannot_change~
            user_requested~
            permissions\n")
        ;
        break;
    }
    option_correct2 = 0;
break;
case 5:
    if((chmod(file_name ,
        S_IRUSR | S_IWUSR)) ==
        -1)
    {
        printf("Error:~
            cannot_change~
            user_requested~
            permissions\n")
        ;
        break;
    }
    option_correct2 = 0;
break;
case 6:
    if((chmod(file_name ,
        S_IRUSR | S_IXUSR)) ==
        -1)
    {
        printf("Error:~
            cannot_change~
            user_requested~
            permissions\n")
        ;
        break;
    }
    option_correct2 = 0;
break;
case 7:
    if((chmod(file_name ,
        S_IWUSR | S_IXUSR)) ==
        -1)
    {
        printf("Error:~
            cannot_change~
            user_requested~
            permissions\n")
        ;
        break;
    }
    option_correct2 = 0;
break;
default: printf("Choose_a_correct_
    option.\n"); break;

```

```

    }
    printf("Done!\n");
}
break;
case 2:
    while(option_correct2 == 1)
    {
        printf("\nWhich group permissions would you
        like to change?\n");
        printf("1: Set read, write and execute
        permissions\n");
        printf("2: Set read permission\n");
        printf("3: Set write permission\n");
        printf("4: Set execute permission\n");
        printf("5: Set read and write permissions\n
        ");
        printf("6: Set read and execute permissions
        \n");
        printf("7: Set write and execute
        permissions\n");
        scanf("%d", &option2);
        switch(option2)
        {
            case 1:
                if((chmod(file_name,
                S_IRWXG)) == -1)
                {
                    printf("Error:
                    cannot change
                    user requested
                    permissions\n");
                    ;
                    break;
                }
                option_correct2 = 0;
            break;
            case 2:
                if((chmod(file_name,
                S_IRGRP)) == -1)
                {
                    printf("Error:
                    cannot change
                    user requested
                    permissions\n");
                    ;
                    break;
                }
                option_correct2 = 0;
            break;
            case 3:
                if((chmod(file_name,
                S_IWGRP)) == -1)
                {
                    printf("Error:
                    cannot change
                    user requested
                    permissions\n");
                    ;
                    break;
                }
                option_correct2 = 0;
            break;
            case 4:
                if((chmod(file_name,
                S_IXGRP)) == -1)
                {
                    printf("Error:
                    cannot change
                    user requested
                    permissions\n");

```

```

                                break;
                                }
                                option_correct2 = 0;
break;
case 5:
    if ((chmod(file_name ,
                S_IRGRP | S_IWGRP)) ==
        -1)
    {
        printf("Error:_
                cannot_change_
                user_requested_
                permissions\n");
        break;
    }
    option_correct2 = 0;
break;
case 6:
    if ((chmod(file_name ,
                S_IRGRP | S_IXGRP)) ==
        -1)
    {
        printf("Error:_
                cannot_change_
                user_requested_
                permissions\n");
        break;
    }
    option_correct2 = 0;
break;
case 7:
    if ((chmod(file_name ,
                S_IWGRP | S_IXGRP)) ==
        -1)
    {
        printf("Error:_
                cannot_change_
                user_requested_
                permissions\n");
        break;
    }
    option_correct2 = 0;
break;
default: printf("Choose_a_correct_
                option.\n"); break;
    }
    printf("Done!\n");
}
break;
case 3:
    while(option_correct2 == 1)
    {
        printf("\nWhich_other_permissions_would_you
                like_to_change?\n");
        printf("1:_Set_read,_write_and_execute_
                permissions\n");
        printf("2:_Set_read_permission\n");
        printf("3:_Set_write_permission\n");
        printf("4:_Set_execute_permission\n");
        printf("5:_Set_read_and_write_permissions\n
                ");
        printf("6:_Set_read_and_execute_permissions
                \n");
        printf("7:_Set_write_and_execute_
                permissions\n");
        scanf("%hd", &option2);
    }
}

```



```

switch(option2)
{
    case 1:
        if ((chmod(file_name ,
                    S_IRWXO)) == -1)
        {
            printf("Error: _
                    cannot _change _
                    user _requested _
                    permissions\n")
            ;
            break;
        }
        option_correct2 = 0;
    break;
    case 2:
        if ((chmod(file_name ,
                    S_IROTH)) == -1)
        {
            printf("Error: _
                    cannot _change _
                    user _requested _
                    permissions\n")
            ;
            break;
        }
        option_correct2 = 0;
    break;
    case 3:
        if ((chmod(file_name ,
                    S_IWOTH)) == -1)
        {
            printf("Error: _
                    cannot _change _
                    user _requested _
                    permissions\n")
            ;
            break;
        }
        option_correct2 = 0;
    break;
    case 4:
        if ((chmod(file_name ,
                    S_IXOTH)) == -1)
        {
            printf("Error: _
                    cannot _change _
                    user _requested _
                    permissions\n")
            ;
            break;
        }
        option_correct2 = 0;
    break;
    case 5:
        if ((chmod(file_name ,
                    S_IROTH | S_IWOTH)) ==
            -1)
        {
            printf("Error: _
                    cannot _change _
                    user _requested _
                    permissions\n")
            ;
            break;
        }
        option_correct2 = 0;
    break;
    case 6:

```

```

        if ((chmod(file_name,
                    S_IROTH | S_IXOTH)) ==
            -1)
        {
            printf("Error: _
                    cannot _change _
                    user _requested _
                    permissions\n")
            ;
            break;
        }
        option_correct2 = 0;
    break;
    case 7:
        if ((chmod(file_name,
                    S_IWOTH | S_IXOTH)) ==
            -1)
        {
            printf("Error: _
                    cannot _change _
                    user _requested _
                    permissions\n")
            ;
            break;
        }
        option_correct2 = 0;
    break;
    default: printf("Choose _a _correct _
                    option.\n"); break;
        }
        printf("Done!\n");
    }
    break;
    case 4:
        exit(EXIT_SUCCESS);
    break;
    default:
        printf("Choose _a _correct _option\n");
    break;
}
}

closedir(directory_stream);

return 0;
}

```

## 2.6. Punto 10 Linux

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.

Al correr el programa lo haremos indicando el nombre del archivo ejecutable y el nombre del directorio que queremos visualizar, al teclear enter nos aparecerá la información solicitada:

```

james@james-Lenovo-ideapad-320-15ABR:~/Documents/ESCOM_SEMESTRE_5/2CM9_SIST
PERATIVOS/1_Parcial/2_Práctica$ ./a.out E
>> Name: c.txt Size: 99 bytes Last file access: Sun Mar 11 14:58:09 2018
>> Name: .. Size: 4096 bytes Last file access: Sun Mar 11 15:14:37 2018
>> Name: . Size: 4096 bytes Last file access: Sun Mar 11 14:59:04 2018
>> Name: d.txt Size: 99 bytes Last file access: Sun Mar 11 14:58:09 2018
>> Name: a.txt Size: 99 bytes Last file access: Sun Mar 11 14:58:09 2018
>> Name: e.txt Size: 99 bytes Last file access: Sun Mar 11 14:58:09 2018
>> Name: b.txt Size: 99 bytes Last file access: Sun Mar 11 14:58:09 2018
james@james-Lenovo-ideapad-320-15ABR:~/Documents/ESCOM_SEMESTRE_5/2CM9_SIST
PERATIVOS/1_Parcial/2_Práctica$

```

Figura 8:

### 2.6.1. Código

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

//EL NO DE ARCHIVOS QUE SE SUPONE HAY COMO MAXIMO EN UN DIRECTORIO
#define NO_FILES 10

int main(int argc, char const *argv[])
{
    DIR *directory_stream;
    struct dirent *filep;
    struct stat file_status[NO_FILES];
    char *file_names[NO_FILES];
    short i = 0, j;

    //ABRIMOS EL DIRECTORIO
    if((directory_stream = opendir(argv[1])) == NULL)
    {
        printf("Error: cannot open the directory.\n");
        exit(EXIT_FAILURE);
    }

    //CAMBIAMOS DE DIRECTORIO PARA PODER USAR NOMBRE RELATIVOS
    if((chdir(argv[1])) == -1)
    {
        printf("Error: cannot change the directory.\n");
        exit(EXIT_FAILURE);
    }

    //GUARDAMOS LOS ESTATUS DE LOS ARCHIVOS DENTRO DEL DIRECTORIO EN UN ARREGLO
    while((filep = readdir(directory_stream)) != NULL)
    {
        if((stat(filep->d_name, &file_status[i])) == -1)
        {
            printf("Error: cannot read file status of file: %s\n",
                filep->d_name);
            exit(EXIT_FAILURE);
        }
        file_names[i++] = filep->d_name;
    }

    //IMPRIMIMOS LA INFORMACION DE CADA ARCHIVO
    for(j = 0; j < i; j++)
        printf(">>Name: %s Size: %ld bytes Last file access: %s",
            file_names[j], (long long) file_status[j].st_size, ctime(&
            file_status[j].st_atime));

    closedir(directory_stream);

    return 0;
}
```

## 2.7. Punto 10 Windows

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

Al correr el programa lo hacemos indicando el nombre del archivo ejecutable y el nombre del directorio que queremos visualizar, al teclear enter nos aparecerá la información solicitada:

```

C:\Users\James\Documents\ESCOM_SEMESTRE_5\2CM9_SISTEMAS_OPERATIVOS\1_Parcial\2_Práctica\Windows>a E
>> Name: e.txt Size: 0 bytes Last file access: Sun Mar 11 18:52:01 2018
>> Name: e.txt Size: 0 bytes Last file access: Sun Mar 11 18:58:03 2018
>> Name: e.txt Size: 99 bytes Last file access: Sun Mar 11 18:52:01 2018
>> Name: e.txt Size: 99 bytes Last file access: Sun Mar 11 18:52:01 2018
>> Name: e.txt Size: 99 bytes Last file access: Sun Mar 11 18:52:01 2018
>> Name: e.txt Size: 99 bytes Last file access: Sun Mar 11 18:52:01 2018
>> Name: e.txt Size: 99 bytes Last file access: Sun Mar 11 18:52:01 2018
>> Name: e.txt Size: 99 bytes Last file access: Sun Mar 11 18:52:01 2018

```

Figura 9:

### 2.7.1. Código

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <Windows.h>
#include <dirent.h>
#include <sys/stat.h>

//EL NO DE ARCHIVOS QUE SE SUPONE HAY COMO MAXIMO EN UN DIRECTORIO
#define NO_FILES 10

int main(int argc, char const *argv[])
{
    DIR *directory_stream;
    struct dirent *filep;
    struct stat file_status[NO_FILES];
    char *file_names[NO_FILES];
    short i = 0, j;

    //ABRIMOS EL DIRECTORIO
    if((directory_stream = opendir(argv[1])) == NULL)
    {
        printf("Error: _cannot_open_the_directory.\n");
        exit(EXIT_FAILURE);
    }

    //CAMBIANDO DE DIRECTORIO
    if((SetCurrentDirectory(argv[1])) == 0)
    {
        printf("Error: _Cannot_change_the_directory.\n");
        exit(EXIT_FAILURE);
    }

    //GUARDAMOS LOS ESTATUS DE LOS ARCHIVOS DENTRO DEL DIRECTORIO EN UN ARREGLO
    while((filep = readdir(directory_stream)) != NULL)
    {
        if((stat(filep->d_name, &file_status[i])) == -1)
        {
            printf("Error: _cannot_read_file_status_of_file:_%s\n",
                filep->d_name);
            exit(EXIT_FAILURE);
        }
        file_names[i++] = filep->d_name;
    }

    //IMPRIMIMOS LA INFORMACION DE CADA ARCHIVO
    for(j = 0; j < i; j++)
        printf(">>_Name:_%s__Size:_%ld_bytes__Last_file_access:_%s",
            file_names[j], (long long) file_status[j].st_size, ctime(&
            file_status[j].st_atime));

    closedir(directory_stream);

    return 0;
}

```

## 2.8. Punto 11 Linux

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 copie uno o más de los archivos creados a un directorio previamente establecido.

Al correr el programa lo hacemos indicando el nombre del archivo ejecutable y el nombre del directorio que queremos visualizar, al teclear enter nos aparecerá un listado con los archivos que contiene el directorio y un menú como el siguiente:

```
james@james-Lenovo-Ideapad-320-15ABR:~/Documents/ESCOM_SEMESTRE_5/2CM9_SISTEMAS
OPERATIVOS/1_Parcial/2_Práctica$ ./a.out /home/james/Documents/ESCOM_SEMESTRE_5/
2CM9_SISTEMAS_OPERATIVOS/1_Parcial/2_Práctica/E
Files in the directory:
>> c.txt
>> ..
>> .
>> d.txt
>> a.txt
>> e.txt
>> b.txt

MENU
1. Show contents of some file.
2. Copy one or more files to another directory (it musts previously exist)
3. Exit
Choose an option: 
```

Figura 10:

Si elegimos la opción 1 nos pedirá el nombre del archivo del cual queremos visualizar sus contenidos y posteriormente nos los mostrará:

```
MENU
1. Show contents of some file.
2. Copy one or more files to another directory (it musts previously exist)
3. Exit
Choose an option: 1
Write the name of the file: a.txt
Contents:
abbbbaabbbbbaaaaaabbaabababbbababbababbabbbaaabababaabaaaabbbbabbabaababaaababaaa
abbbababbabbabbbbbaa
```

Figura 11:

Si elegimos la opción 2 nos pedirá el nombre del directorio al cual queremos copiar los archivos, el número de archivos a copiar y el nombre de estos archivos:

```
MENU
1. Show contents of some file.
2. Copy one or more files to another directory (it musts previously exist)
3. Exit
Choose an option: 2
Enter the directory you want to paste the files: /home/james/Documents/ESCOM_SEMESTRE_5/2CM9_SISTEMAS_OPERATIVOS/1_Parcial/2_Práctica/A
Enter the number of files you want to copy: 2
Enter the name of the 1 file: a.txt
Enter the name of the 2 file: b.txt
```

Figura 12:

### 2.8.1. Código

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <unistd.h>
#include <fcntl.h>

int main(int argc, char *argv[])
{
    DIR *directory_stream;
    struct dirent *file_pointer;
    short option, no_files, i, j = 0;
    char file_name[FILENAME_MAX], *file_buf, new_directory[FILENAME_MAX], **
        file_cnames, old_directory[FILENAME_MAX];
    int file_descriptor, old_directory_len;
    long long file_len = 0;

    //ABRIMOS EL DIRECTORIO
    if((directory_stream = opendir(argv[1])) == NULL)
    {
        printf("Error: cannot open the directory.\n");
        exit(EXIT_FAILURE);
    }

    //CAMBIAMOS DE DIRECTORIO
    if((chdir(argv[1])) == -1)
    {
        printf("Error: cannot change the directory.\n");
        exit(EXIT_FAILURE);
    }

    //MOSTRAMOS EL NOMBRE DE LOS ARCHIVOS DENTRO DEL DIRECTORIO
    printf("Files in the directory:\n");
    while((file_pointer = readdir(directory_stream)) != NULL)
        printf(">> %s\n", file_pointer->d_name);

    for(;;)
    {
        printf("\n\nMENU\n");
        printf("1. Show contents of some file.\n");
        printf("2. Copy one or more files to another directory (it musts_
            previously exist)\n");
        printf("3. Exit\n");
        printf("Choose an option:");
        scanf("%d", &option);
        switch(option)
        {
            case 1:
                printf("Write the name of the file:");
                scanf("%s", file_name);
                if((file_descriptor = open(file_name, O_RDONLY)) ==
                    -1)
                {
                    printf("Error: cannot open the file.\n");
                    exit(EXIT_FAILURE);
                }
                printf("Contents:\n");
                file_buf = malloc(2);
                while((read(file_descriptor, file_buf, 1)) != 0)
                    putchar(file_buf[0]);

            break;
            case 2:
                printf("Enter the directory you want to paste the_
                    files:");
                scanf("%s", new_directory);
                printf("Enter the number of files you want to copy:
                    \n");

```

```

scanf("%d", &no_files);
file_cnames = malloc(no_files * sizeof(char *));
    //asignando espacio para el n mero de nombres
    de archivos a guardar
//PEDIMOS EL NOMBRE DE CADA ARCHIVO Y LOS CREAMOS
    EN EL NUEVO DIRECTORIO
for(i = 0; i < no_files; i++)
{
    //ASIGNANDO ESPACIO PARA EL NOMBRE DE CADA
    ARCHIVO
    file_cnames[i] = malloc(FILENAME_MAX);
    printf("Enter the name of the %d file: ",
        i + 1);
    scanf("%s", file_cnames[i]);

    //CAMBIAMOS AL VIEJO DIRECTORIO
    if((chdir(argv[1])) == -1)
    {
        printf("Error: failed to change to
            old_directory.\n");
        exit(EXIT_FAILURE);
    }

    //ABRIMOS EL VIEJO ARCHIVO
    if((file_descriptor = open(file_cnames[i],
        ORDONLY)) == -1)
    {
        printf("Error: failed to open %s\n",
            file_cnames[i]);
        exit(EXIT_FAILURE);
    }

    /*
    //CALCULAMOS EL N MERO DE CARACTERES EN EL
    ARCHIVO VIEJO
    while((read(file_descriptor, file_buf, 1))
        != 0)
        file_len++;

    //NOS COLOCAMOS DE NUEVO EN EL INCIO DEL
    ARCHIVO
    lseek(file_descriptor, 0, SEEK_SET);
    */

    //ASIGNAMOS ESPACIO EN EL BUFFER DE ACUERDO
    AL TAMA O DEL ARCHIVO
    file_buf = malloc(100);

    //LEEMOS TODO EL ARCHIVO Y LO GUARDAMOS EN
    FILE_BUF
    while((read(file_descriptor, file_buf, 90))
        != 0)
        ;

    //CERRAMOS EL VIEJO ARCHIVO
    close(file_descriptor);

    //CAMBIAMOS AL NUEVO DIRECTORIO
    if((chdir(new_directory)) == -1)
    {
        printf("Error: failed to change to
            new_directory.\n");
        exit(EXIT_FAILURE);
    }

    //CREANDO EL ARCHIVO CON EL NOMBRE
    ESPECIFICADO
    if((file_descriptor = creat(file_cnames[i],
        S_IRWXU)) == -1)
    {

```





```

MENU
1. Show contents of some file.
2. Copy one or more files to another directory (it musts previously exist)
3. Exit
Choose an option: 2
Enter the directory you want to paste the files: C:\Users\James\Documents\ESCOM_SEMESTRE_5\2CM9_SIS
rcial\2_Práctica\Windows\A
Enter the number of files you want to copy: 2
Enter the name of the 1 file: a.txt
C:\Users\James\Documents\ESCOM_SEMESTRE_5\2CM9_SISTEMAS_OPERATIVOS\1_Parcial\2_Práctica\Windows\A
Error: Cannot change the directory.

```

Figura 15:

### 2.9.1. Código

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <Windows.h>
#include <dirent.h>
#include <sys/stat.h>

int main(int argc, char *argv[])
{
    DIR *directory_stream;
    struct dirent *file_pointer;
    short option, no_cfiles, i, j = 0;
    char file_name[FILENAME_MAX], *file_buf, new_directory[FILENAME_MAX], **
        file_cnames, old_directory[FILENAME_MAX];
    HANDLE file_descriptor;
    long long file_len = 0;

    //ABRIMOS EL DIRECTORIO
    if((directory_stream = opendir(argv[1])) == NULL)
    {
        printf("Error: cannot open the directory.\n");
        exit(EXIT_FAILURE);
    }

    //CAMBIANDO DE DIRECTORIO
    if((SetCurrentDirectory(argv[1])) == 0)
    {
        printf("Error: Cannot change the directory.\n");
        exit(EXIT_FAILURE);
    }

    //MOSTRAMOS EL NOMBRE DE LOS ARCHIVOS DENTRO DEL DIRECTORIO
    printf("Files in the directory: \n");
    while((file_pointer = readdir(directory_stream)) != NULL)
        printf(">> %s\n", file_pointer->d_name);

    for(;;)
    {
        printf("\n\nMENU\n");
        printf("1. Show contents of some file.\n");
        printf("2. Copy one or more files to another directory (it musts\n
            previously exist)\n");
        printf("3. Exit\n");
        printf("Choose an option: ");
        scanf("%d", &option);
        switch(option)
        {
            case 1:
                printf("Write the name of the file: ");
                scanf("%s", file_name);
                if((file_descriptor = CreateFile(file_name,
                    GENERIC_READ, FILE_SHARE_READ, NULL,
                    OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL)) ==
                    INVALID_HANDLE_VALUE)
                {
                    printf("Error: cannot open the file.\n");

```

```

        exit(EXIT_FAILURE);
    }
    printf("Contents: \n");
    file_buf = malloc(2);
    if((ReadFile(file_descriptor, file_buf, 98, NULL,
        NULL)) == 0)
    {
        printf("Error: _failed_to_read_the_file.\n");
        ;
        exit(EXIT_FAILURE);
    }
    printf("%s\n", file_buf);
    CloseHandle(file_descriptor);
break;
case 2:
    printf("Enter the directory you want to paste the
        files: ");
    scanf("%s", new_directory);
    printf("Enter the number of files you want to copy:
        ");
    scanf("%d", &no_files);
    file_cnames = malloc(no_files * sizeof(char *));
        //asignando espacio para el n mero de nombres
        de archivos a guardar
    //PEDIMOS EL NOMBRE DE CADA ARCHIVO Y LOS CREAMOS
    EN EL NUEVO DIRECTORIO
    for(i = 0; i < no_files; i++)
    {
        //ASIGNANDO ESPACIO PARA EL NOMBRE DE CADA
        ARCHIVO
        file_cnames[i] = malloc(FILENAME_MAX);
        printf("Enter the name of the %d file: ",
            i + 1);
        scanf("%s", file_cnames[i]);

        //CAMBIAMOS AL VIEJO DIRECTORIO
        if((SetCurrentDirectory(argv[1])) == 0)
        {
            printf("Error: _Cannot_change_the_
                directory.\n");
            exit(EXIT_FAILURE);
        }

        //ABRIMOS EL VIEJO ARCHIVO
        if((file_descriptor = CreateFile(
            file_cnames[i], GENERIC_READ,
            FILE_SHARE_READ, NULL, OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL, NULL)) ==
            INVALID_HANDLE_VALUE)
        {
            printf("Error: _cannot_open_the_file
                .\n");
            exit(EXIT_FAILURE);
        }

        /*
        //CALCULAMOS EL N MERO DE CARACTERES EN EL
        ARCHIVO VIEJO
        while((read(file_descriptor, file_buf, 1))
            != 0)
            file_len++;

        //NOS COLOCAMOS DE NUEVO EN EL INCIO DEL
        ARCHIVO
        lseek(file_descriptor, 0, SEEK_SET);
        */

        //ASIGNAMOS ESPACIO EN EL BUFFER DE ACUERDO
        AL TAMA O DEL ARCHIVO
        file_buf = malloc(100);

```

```

//LEEMOS TODO EL ARCHIVO Y LO GUARDAMOS EN
FILE_BUF
if((ReadFile(file_descriptor , file_buf , 90,
NULL, NULL)) == 0)
{
    printf("Error: _failed_to_read_the_
file.\n");
    exit(EXIT_FAILURE);
}

//CERRAMOS EL VIEJO ARCHIVO
CloseHandle(file_descriptor);

printf("%s\n", new_directory);
//CAMBIAMOS AL NUEVO DIRECTORIO
if((SetCurrentDirectory(new_directory)) ==
0)
{
    printf("Error: _Cannot_change_the_
directory.\n");
    exit(EXIT_FAILURE);
}

//CREANDO EL ARCHIVO CON EL NOMBRE
ESPECIFICADO
if((file_descriptor = CreateFile(
file_descriptor , GENERIC_READ |
GENERIC_WRITE, FILE_SHARE_WRITE, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL,
NULL)) == INVALID_HANDLE_VALUE)
{
    printf("Error: _Cannot_create_the_
file.\n");
    exit(EXIT_FAILURE);
}

//ESCRIBIMOS HACIA EL NUEVO ARCHIVO EL
CONTENIDO DEL VIEJO
WriteFile(file_descriptor , file_buf , 90,
NULL, NULL);

//CERRAMOS EL NUEVO ARCHIVO
CloseHandle(file_descriptor);
}
break;
case 3:
    exit(EXIT_SUCCESS);
break;
default:
    printf("Choose_a_correct_option.\n");
break;
}
return 0;
}
}

```

### **3. Análisis Crítico**

Esta práctica fue mucho más complicada que la anterior para mí ya que estaba poco familiarizado con el sistema operativo Linux y tuve que investigar por muchas otras partes varios de los detalles que en clase no se ven y son necesarios para poder desarrollar los programas.

### **4. Observaciones**

En el sistema operativo Linux es relativamente sencillo trabajar con las llamadas al sistema y más si se trata de desarrollo de aplicaciones con el lenguaje C, ya que se siente como todo parte de un mismo ambiente de trabajo.

### **5. Conclusión**

Esta práctica fue un reto aún más grande que la primera pero proporcionalmente más educativa, aprendí muchas cosas no solo sobre el sistema Linux con el cual no estaba familiarizado sino también con Windows del cual a pesar de haberlo usado toda mi vida me di cuenta no conocía ni la punta del iceberg.