**Práctica 2. Introducción a los sistemas operativos Linux y Windows (2)**

**Grupo: 2CM9**

**Equipo:**

* **Guerra Vargas Irving Cristóbal**
* **Reyes Valenzuela Alejandro**
* **Dávila Méndez Juan Manuel**
* **Hernandez Patrocinio Luis Enrique**

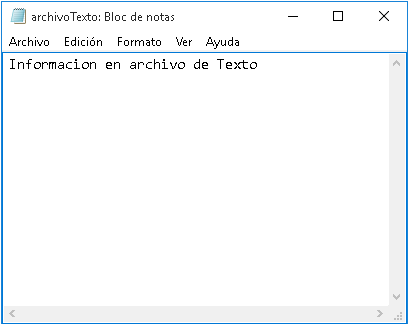
**Competencia.**

* **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.**

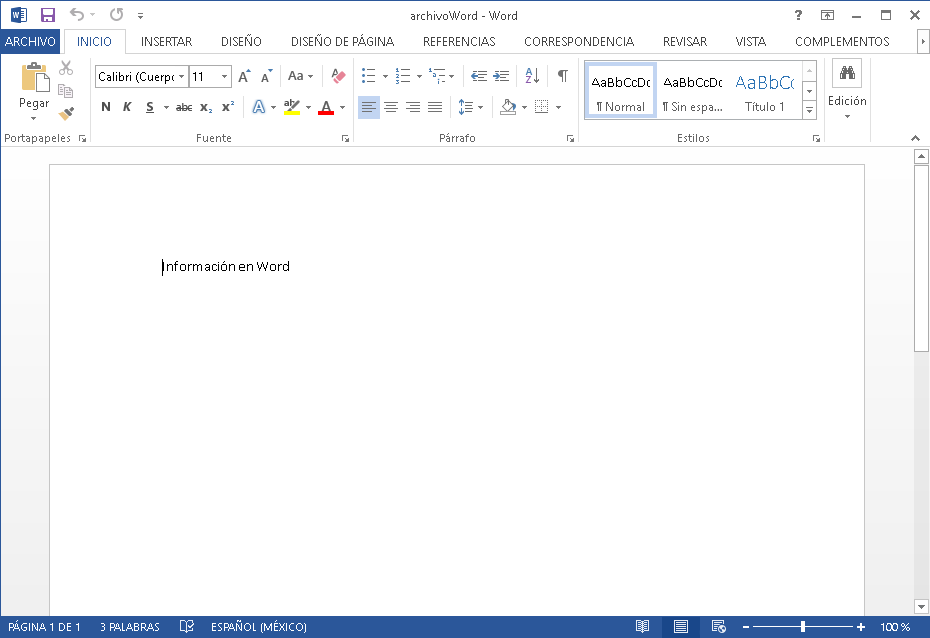
**Desarrollo.**

**1.- 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 de Texto - Notepad :**

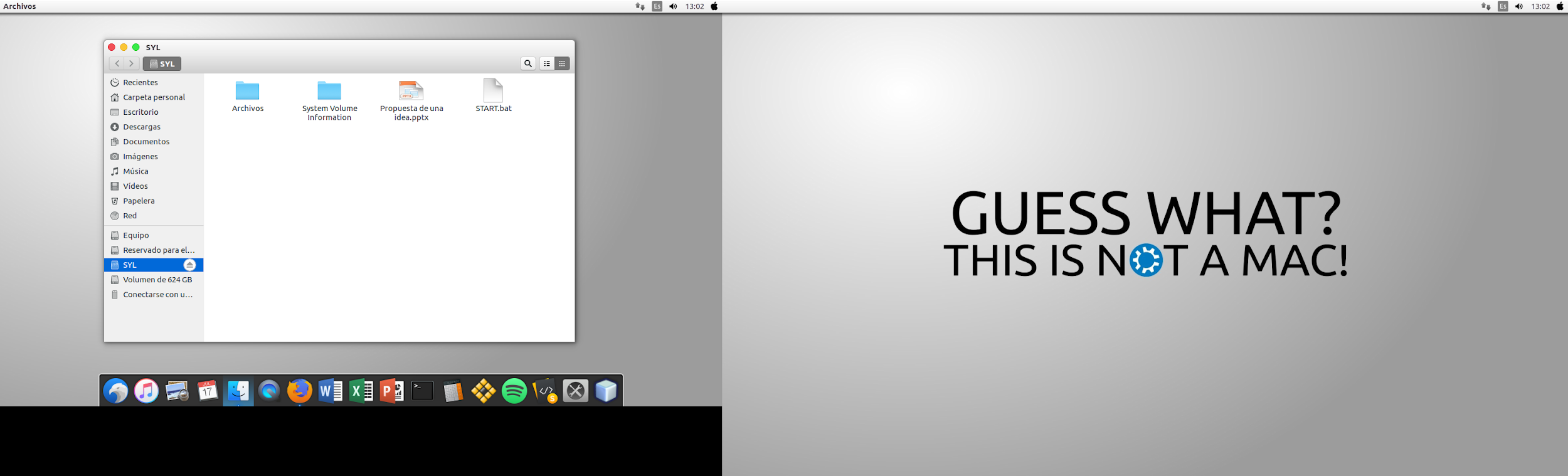


**Archivo de Word:**

****

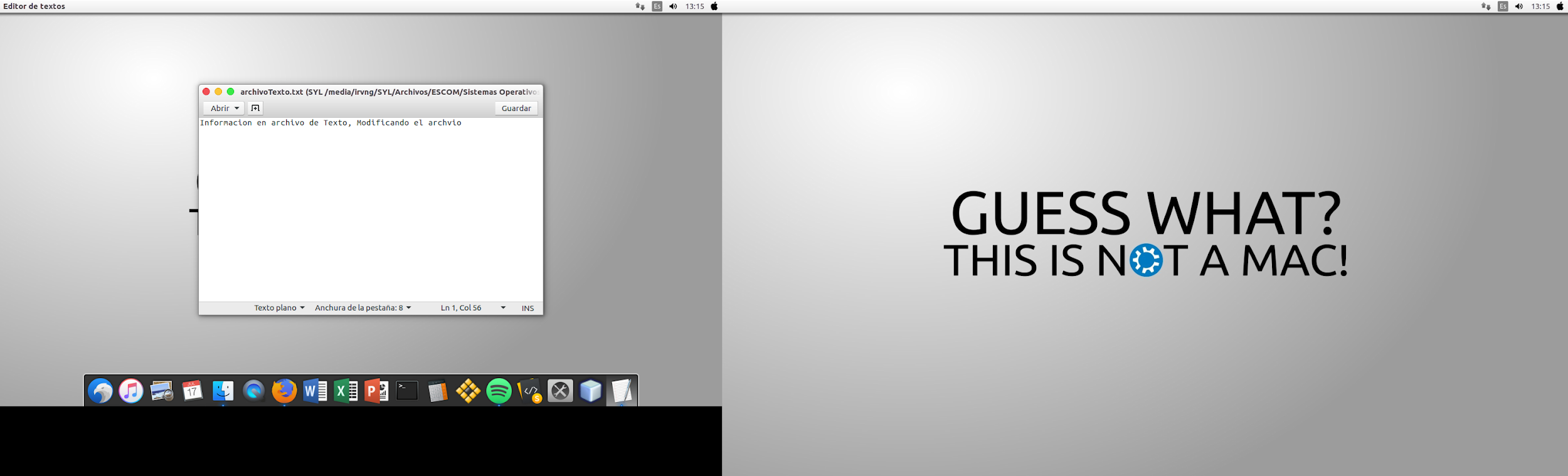
**2.- Inicie sesión en Linux**

**3.- Verifique si está montada la unidad de memoria usb en su sistema, para ello introduzca la memoria usb y observe si es reconocida en el escritorio.**

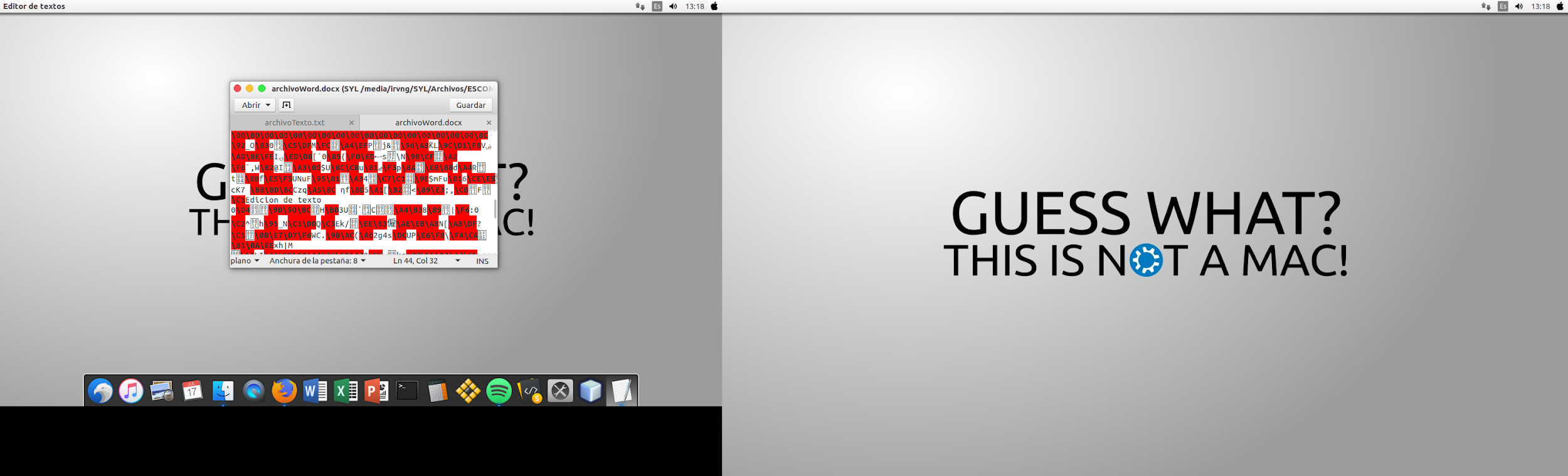
****

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

**Archivo de Texto:**

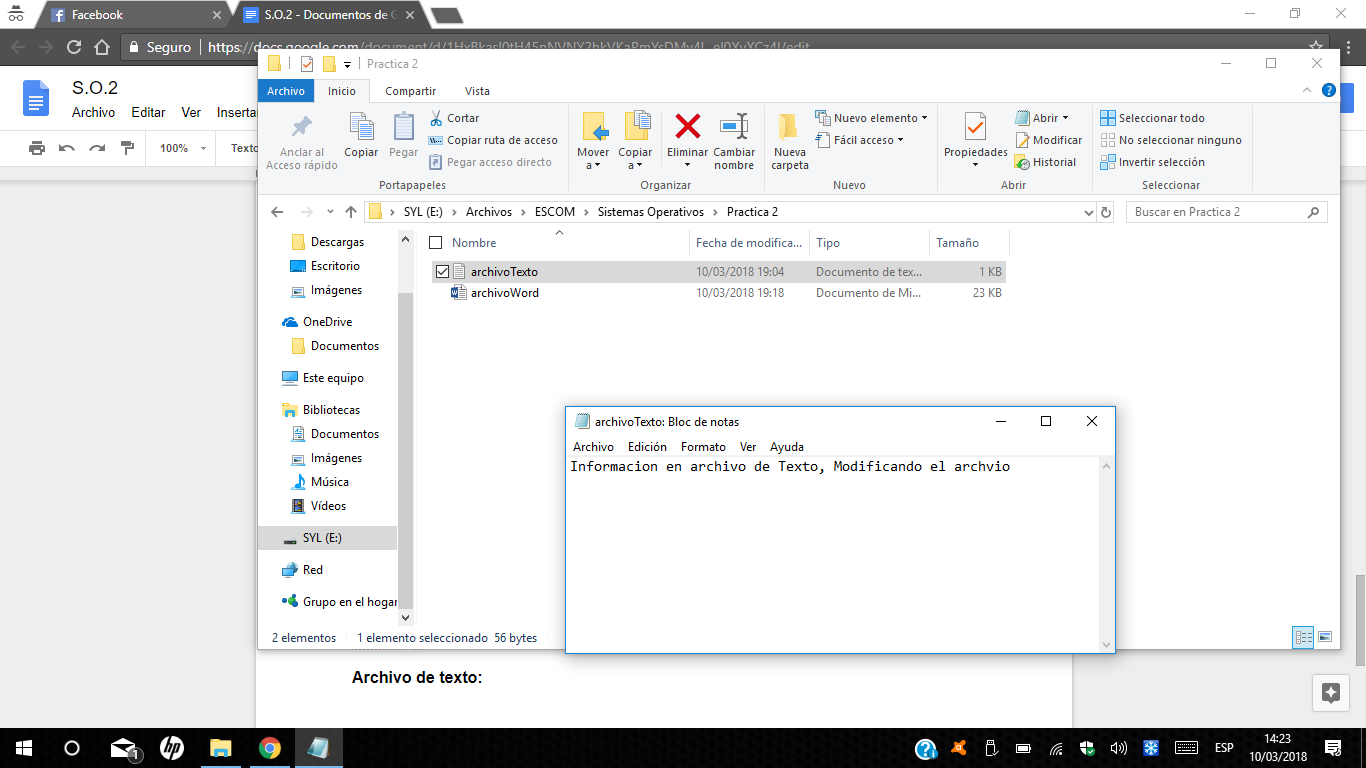
****

**Archivo de Word: (Prácticamente imposible editarlo, sin saber que cambiaŕa)**

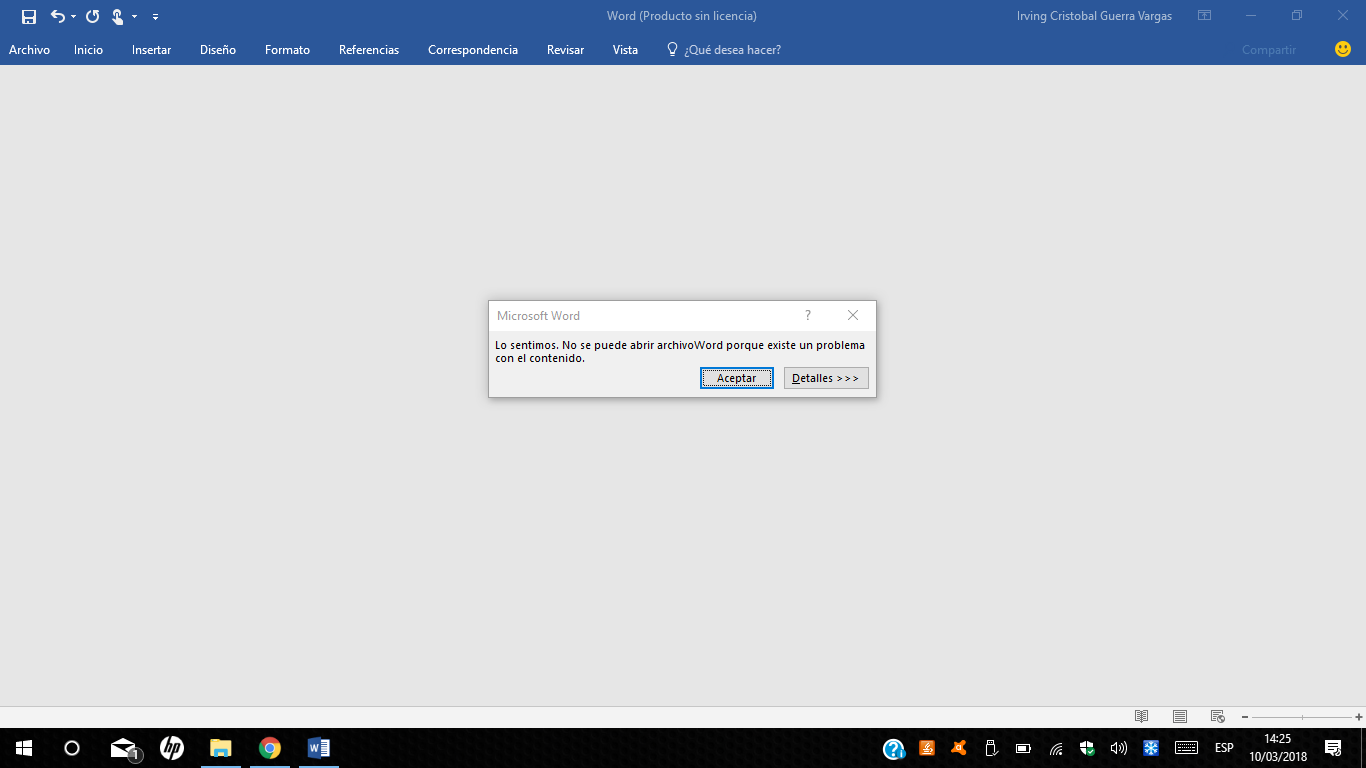
****

**5.- Inicie sesión en Windows y observe el contenido de sus archivos en su memoria USB. ¿Se observan modificaciones realizadas en Linux?, explique el porqué si o no se observan.**

**Archivo de texto: Si, ya que tanto windows como ubuntu pueden abrir archivos Txt, y no importa, mientras tengan un editor de txt podrán modificar y visualizar la misma información.**

****

**Archivo de Word: NO, ya que Word es un software únicamente de Windows, de la misma compañía Microsoft, y al abrir una archivo word en ubuntu con editor de textos, cambia todo, ahora word ni siquiera puede abrirlo. La solución es usar en ubuntu un programa que abra documentos de word como LibreOffice.**

****

**6.- A través de su manual en línea de Linux (man) en la segunda o tercera sección, investigue y reporte para que sirven las siguientes llamadas al sistema de Linux:**

|  |  |
| --- | --- |
| **Llamada al Sistema** | **Descripción / Uso** |
| **open** | Sirve para abrir un archivo, o crearlo, y dependiendo las circunstancias, en modo de lectura, escritura, entre otros modos. Su valor de retorno es un número entero, mismo que es un descriptor de archivo. |
| **close** | Sirve para cerrar un descriptor de archivo, previamente abierto por Open. |
| **read** | Intenta leer, dada la cantidad de bytes, de un archivo previamente abierto por Open(). Su valor de retorno es la cantidad de bytes leídos, al terminar de leer, regresa 0. |
| **write** | Escribe a un descriptor de archivo, utilizando como datos de entrada un búfer, que puede contener cualquier cosa y una cantidad de bytes a escribir. |
| **creat** | Sirve para abrir (o crear) archivos. Open() incluye ya la funcionalidad de esta llamada al sistema, pues también es capaz de crear archivos. |
| **Iseek** | Sirve para ubicarse, en cierta posición dentro de un archivo. Su valor de retorno es el valor de ubicación medido en bytes desde la ubicación del archivo. |
| **access** | Checa si el proceso en cuestión puede acceder a cierto archivo. También sirve para verificar qué permisos se tienen sobre el archivo. Su valor de retorno es 0 en caso de que el acceso sea exitoso. |
| **stat** | Se utiliza para obtener datos de archivos. Se usa en conjunto con una estructura de tipo stat, en la cual se almacenarán los detalles del archivo accedido en cuestión. Su valor de retorno es 0 si es que se pudo abrir el archivo correctamente. |
| **chmod** | Se utiliza para cambiar los permisos en un archivo. Recibe como parámetros el nombre del archivo en cuestión y los permisos (en decimal) a establecer. |
| **chown** | Cambia la propiedad (es decir, a qué grupo y usuario) de un archivo. Su valor de retorno es 0 en caso de que el cambio sea exitoso. |
| **fcntl** | Modifica las propiedades de un descriptor de archivo. Se utiliza principalmente para bloquear archivos, acción que permite que sólo un programa acceda a un archivo a la vez. |
| **chdir** | Cambia el directorio sobre el cual se está trabajando al momento de ejecutar el programa. Prácticamente es como el comando cd, pero con llamadas al sistema. |
| **mkdir** | Se utiliza para crear directorios. Sus argumentos de entrada son el nombre (o path) del directorio a crear y el modo, o permisos con el que será creado. |
| **opendir** | Abre un directorio. En caso de abrirlo correctamente, su valor de retorno es un apuntador hacia la primera entrada en el directorio. |
| **readdir** | Sirve para leer un directorio y sus contenidos. Su valor de retorno es un apuntador hacia el siguiente entrada dentro del directorio. |

**7.- A traves del sitio MSDN de Microsoft, investigue y reporte para que sirven las siguientes llamadas al sistema de Windows:**

|  |  |
| --- | --- |
| **Llamada al Sistema** | **Descripción / Uso** |
| **OpenFile** | Crea, abre, reabre o elimina un archivo. Recibe tres parámetros:   * IpFileName [in], el nombre del archivo. * IpReOpenBuff [out], un puntero a la estructura OFSTRUC que recibe información sobre el archivo cuando se abre por primera vez. * Style[in], la acción que se va a realizar. Algunas opciones que puede realizar son OF\_CANCEL, OF\_CREATE, OF\_CREATE, OF\_EXIST, etc. |
| **CloseFile** | Cierra un archivo (objeto abierto), recibe un solo parámetro:   * hObject [in]: Identificador del objeto   Regresa un cero si ha ocurrido un error, en caso contrario, regresará otro valor distinto. |
| **ReadFile** | Lee el contenido de un archivo, recibe tres parámetros obligatorios y uno opcional:   * hFile [in]: Identificador del archivo * lpBuffer [out]: Apuntador al buffer que recibirá los datos a leer * NumberOfBytesToRead [in] : Número máximo de bits a leer * lpNumberOfBytesRead [out, opcional] : Apuntador que recibirá el número de bytes leídos.   Regresa un cero si ha ocurrido un error, en caso contrario, regresará otro valor distinto. |
| **WriteFile** | Escribe datos a un dispositivo específico, recibe tres parámetros obligatorios y dos opcionales:   * hFile [in]: Identificador del archivo * lpBuffer [in]: Apuntador al buffer que recibirá los datos a escribir * nNumberOfBytesToWrite [in]: Número de bytes a escribir * lpNumberOfBytesWritten [out, opcional]: Apuntador a variable que registrará el número de bytes escritos * lpOverlapped [in, out, opcional]: Registra una escritura traslapada   Regresa un cero si ha ocurrido un error, en caso contrario, regresará otro valor distinto. |
| **CreateFile** | Crea o abre un archivo, recibe cinco parámetros obligatorios y dos opcionales:   * lpFileName [in]: Identificador del archivo * dwDesiredAccess [in]: Se especifica el tipo de acceso que se realizará, de lectura, escritura, ambas o ninguna * dwShareMode [in]: Se específica cómo será compartido el archivo/dispositivo, el cual puede ser para realizar operaciones de lectura, escritura, ambas, eliminación o ninguna * lpSecurityAttributes [in, opcional]: Apuntador a una estructura que contiene atributos de seguridad * dwCreationDisposition [in]: Específica qué hacer en caso de que el archivo exista o no * dwFlagsAndAttributes [in]: Las banderas y atributos del archivo/dispositivo * hTemplateFile [in, opcional]: identificador válido para un archivo de plantilla con el derecho de acceso GENERIC\_READ.   Regresa el archivo o en caso de error regresa INVALID\_HANDLE\_VALUE. |
| **SetFilePointer** | Mueve un apuntador hacia un archivo en específico, recibe tres parámetros obligatorios y uno opcional:   * hFile [in]: Identificador del archivo * lDistanceToMove [in]: Distancia a la que moveremos el apuntador (Menor a 32 bits) * IpDistanceToMoveHigh [in, out, opcional]: Apundator a los 32 bits más significativos de la distancia de 64 bits para mover * dwMoveMethod [in]: Punto de inicio del apuntador   Regresa DWORD cuando se realizó de manera correcta, en caso de error regresa INVALID\_SET\_FILE\_POINTER. |
| **CreateDirectory** | Crea un nuevo directorio, recibe un parámetro obligatorio y uno opcional:   * lpPathName [in]: Ruta donde se creará el directorio * lpSecurityAttributes [in, opcional]: Apuntador a una estructura que contiene atributos de seguridad   Regresa un cero si ha ocurrido un error, en caso contrario, regresará otro valor distinto. |
| **SetCurrentDirectory** | Cambia el directorio en donde se realizará algún proceso en específico, recibe un sólo parámetro:   * lpPathName [in]: Ruta del nuevo directorio   Regresa un cero si ha ocurrido un error, en caso contrario, regresará otro valor distinto. |

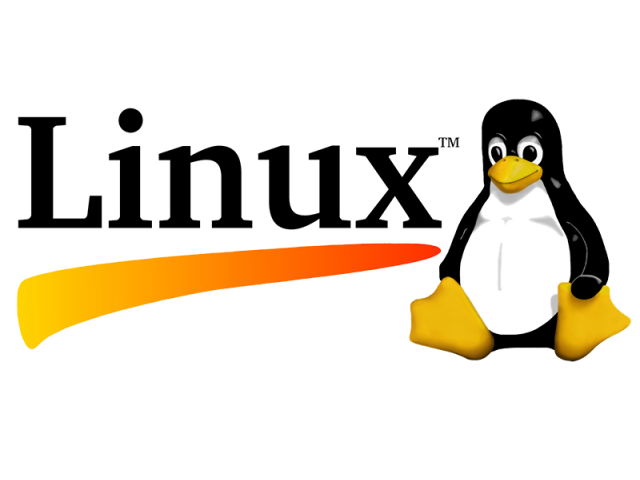
**7.1.- A través del sitio OpenGroup, investigue y reporte para que sirven las siguientes funciones en Windows.**

|  |  |
| --- | --- |
| **Funcion** | **Descripción / Uso** |
| **stat** | Obtiene información acerca de algún archivo en específico, recibe dos parámetros:   * char \*restrict path: Apuntador al archivo * char \*restrict buf: Apuntador al buffer donde se escribirán los datos recabados del archivo |
| **opendir** | Abre un directorio, recibe un parámetro:   * const char \*dirname: Apuntador al directorio a abrir |
| **readdir** | Lee un directorio, recibe una estructura que a su vez recibe como parámetro un apuntador al directorio a leer. |

**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.**

Existen dos llamadas que tienen características similares a chmod, sin embargo una es bastante más aproximada que la otra, la original es -attrib, que normalmente nos ajusta el contenido de un archivo (de público a oculto y viceversa) , sin embargo, no realiza tantas funciones como icalcs, que establece / restablece las listas de control de acceso, por lo que puede otorgar / denegar derechos para SID y grupos individuales

**SECCIÓN LINUX**

****

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

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <dirent.h>**

**#include <time.h>**

**#include <unistd.h>**

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <fcntl.h>**

**#define S\_MAX 20**

**int main(int argc, char \*argv[])**

**{**

**const char \*arreglo\_nice[] = {"a1", "bb2", "ccc3", "dddd4", "5eeeee", "6ffffff", "7ggggggg"}; int arch\_creados=0;**

**if (argv[1]==NULL)**

**{**

**printf("[ERROR] Uso: ./p1 (ruta del directorio deseado)\n");**

**exit(-1);**

**}**

**else**

**{**

**DIR\* dir = opendir(argv[1]); //Se verifica si ya existe el directorio**

**if (dir)**

**{**

**printf("[ERROR] Directorio existente\n"); //Salir si ya existe el directorio**

**closedir(dir);**

**exit(-1);**

**}**

**else**

**{**

**printf("[ OK ] creando nuevo directorio \n");**

**if(mkdir(argv[1], 0755)==0) //Crea el directorio con el path del primer argumento con permisos 0755**

**{**

**printf("[ OK ] directorio %s creado exitosamente\n",argv[1]);**

**chdir(argv[1]); //Entra al directorio creado**

**printf("[ INFO] cantidad maxima de archivos aleatorios: %d\n",S\_MAX);**

**srand(time(NULL)); // hace aleatorios los numeros**

**int alea[S\_MAX], s, randIndexArregloNice, i, tamArregloNice; char nombre[15], buffer[5];**

**tamArregloNice = sizeof(arreglo\_nice) / (2\*sizeof(int)); // Obtiene la cantidad de cadenas presentes en arreglo\_nice**

**for(i=0;i<S\_MAX;i++)**

**{**

**nombre[0] = '\0'; //Inicializa una cadena para contener el nombre**

**strcat(nombre,"archivo"); //Pega la palabra "archivo" a el nombre**

**alea[i] = rand()%S\_MAX; //Obtiene un valor aleatorio entre 0 y S\_MAX**

**sprintf(buffer, "%d", alea[i]); // convierte el numero a cadena para utilizar**

**strcat(nombre,buffer); //Pega el numero aleatorio al final del archivo**

**if(access(nombre, F\_OK) != -1) { //esta funcion se utiliza para obtener el numero de archivos que**

**arch\_creados+=1; //ya existen y se han sobreescrito, para despues imprimir**

**} //la cantidad de archivos aleatorios totales escritos.**

**creat(nombre,0755); //crea nuevos archivos, vacios, con permisos 0755**

**randIndexArregloNice=rand()%tamArregloNice; //Obtiene un elemento aleatoriamente del arreglo\_nice**

**s=open(nombre, O\_WRONLY | O\_APPEND); //Descriptor del archivo a escribir**

**if(write(s, arreglo\_nice[randIndexArregloNice], strlen(arreglo\_nice[randIndexArregloNice]))==-1) //Escribe el elemento aleatorio del arreglo a un archivo**

**printf("[ERROR] Imposible escribir a los archivos creados.\n");**

**close(s); //Cierra el descriptor del archivo**

**}**

**printf("[ INFO] cantidad de archivos creados: %d\n",(S\_MAX-arch\_creados));**

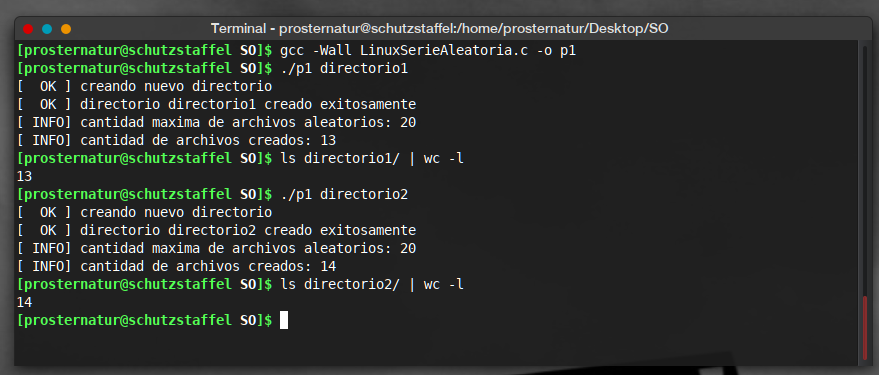
**}**

**else {**

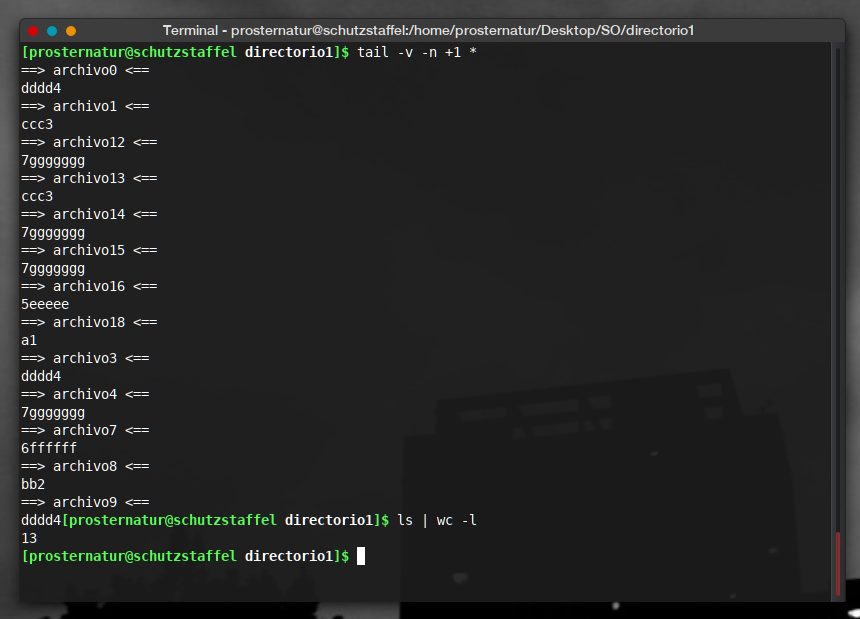
**printf("[ERROR] Algo sucedio mal al crear el directorio.\n ");**

**exit(-1);**

**} } } } // Los corchetes se agruparon para dar espacio a la captura mostrada abajo**

****

Compilación y ejecución: Con el comando *ls | wc -l* se verifica que la cantidad de archivos escritos sea la mostrada en el programa.

****

****

Captura 1: Verificación de que las cadenas estén, en efecto, presentes en cada archivo creado con el comando tail.

Captura 2: Verificación de que las cadenas estén en un archivo con un editor de textos.

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

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <dirent.h>**

**#include <unistd.h>**

**#include <sys/stat.h>**

**int main(int argc, char \*argv[])**

**{**

**char nombre, buffer[100], permisos; int permisos\_int;**

**if (argv[1]==NULL)**

**{**

**printf("[ERROR] Uso: ./p2 (ruta donde se encuentra el archivo a modificar)\n");**

**exit(-1);**

**}**

**else**

**{**

**DIR\* carpeta = opendir(argv[1]); //Se verifica si ya existe el directorio**

**if (carpeta)**

**{**

**printf("[ OK ] Directorio encontrado\n"); //Salir si ya existe el directorio**

**DIR \*d;**

**struct dirent \*carpeta;**

**d = opendir(argv[1]); //Para abrir el directorio en el que se encuentra**

**if (d) {**

**while ((carpeta = readdir(d)) != NULL) { //lista todo hasta que sea el 'fin' de la carpeta**

**if (carpeta->d\_type == DT\_REG) //omite cualquier cosa que no sea un archivo regular**

**printf("%s\n", carpeta->d\_name); //imprime todos los archivos dentro de la carpeta**

**}**

**closedir(d);**

**printf("Ingrese el nombre de cualquier archivo anteriormente mostrado para cambiar permisos: \n");**

**scanf("%s",&nombre);**

**sprintf(buffer, "%s", &nombre); // convierte el numero a cadena para poder usar strtol (convertir a octal)**

**printf("Ingrese permisos en octal para cambiar: \n");**

**scanf("%s",&permisos);**

**permisos\_int = strtol(&permisos, 0, 8); //convierte de octal a decimal**

**printf("[ INFO] Permisos en decimal: %d\n",permisos\_int);**

**chdir(argv[1]); //cambia al directorio deseado**

**if(chmod(buffer,permisos\_int)==0) //chmod lee los permisos en decimal, conversion necesaria**

**printf("[ OK ] Permisos cambiados correctamente.\n");**

**else**

**printf("[ERROR] No se han podido cambiar los permisos.\n");**

**}// gcc -Wall CambiarPermisos.c -o p2**

**}**

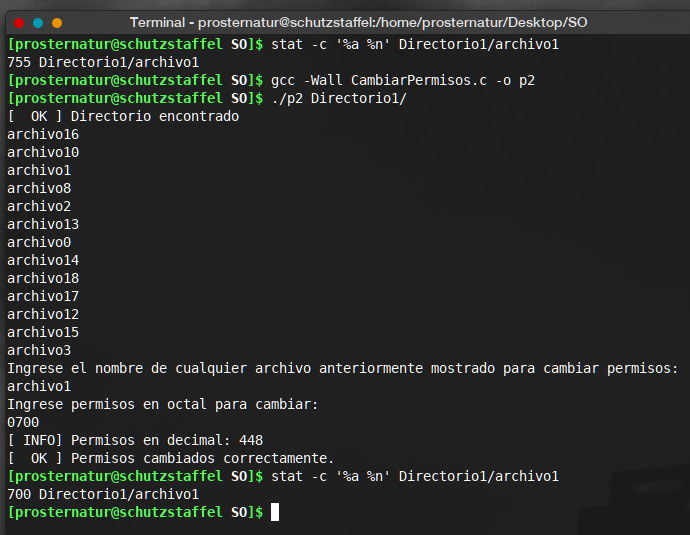
**else**

**printf("[ERROR] No existe el directorio %s\n", argv[1]);**

**}**

**} //stat -c '%a %n' carpetadestino/\***

Captura de pantalla del programa que cambia los permisos de un archivo:

****

Captura 1: Primero se verificó qué permisos tiene archivo1 con el comando *stat*, después se compiló y ejecutó el programa. Posteriormente, se verificó nuevamente que los cambios fueran reales con el comando *stat*.

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

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <sys/types.h>**

**#include <time.h>**

**#include <dirent.h>**

**#include <unistd.h>**

**#include <sys/stat.h>**

**int main(int argc, char \*argv[])**

**{**

**struct stat archivillo;**

**if (argv[1]==NULL)**

**{**

**printf("[ERROR] Uso: ./p3 (ruta donde se encuentran los archivos a revisar\n");**

**exit(-1);**

**}**

**else**

**{**

**DIR\* dir = opendir(argv[1]); //Se verifica si ya existe el directorio**

**if (dir)**

**{**

**printf("[ OK ] Directorio %s encontrado\n", argv[1]); //Salir si ya existe el directorio**

**chdir(argv[1]);**

**DIR \*d;**

**struct dirent \*dir; //la estructura dirent proporciona datos acerca de los archivos**

**d = opendir("."); //Para abrir el directorio en el que se encuentra**

**if (d) {**

**while ((dir = readdir(d)) != NULL) { //lista todo hasta que sea el 'fin' de la carpeta**

**if (dir->d\_type == DT\_REG) {//omite cualquier cosa que no sea un archivo regular**

**printf("%s ", dir->d\_name); //imprime todos los archivos dentro de la carpeta**

**if(stat(dir->d\_name, &archivillo)==0) //abrir cada archivo que se encuentre para mostrar sus atributos**

**printf(" - - - Tam(Bytes): %ld, Fecha: %s", archivillo.st\_size, ctime(&archivillo.st\_atim.tv\_sec)); //convertir la fecha a formato legible**

**else**

**printf("[ERROR] No se pudo leer un archivo.\n");**

**}**

**}**

**closedir(d);**

**}// gcc -Wall ListadorDeArchivos.c -o p3**

**}**

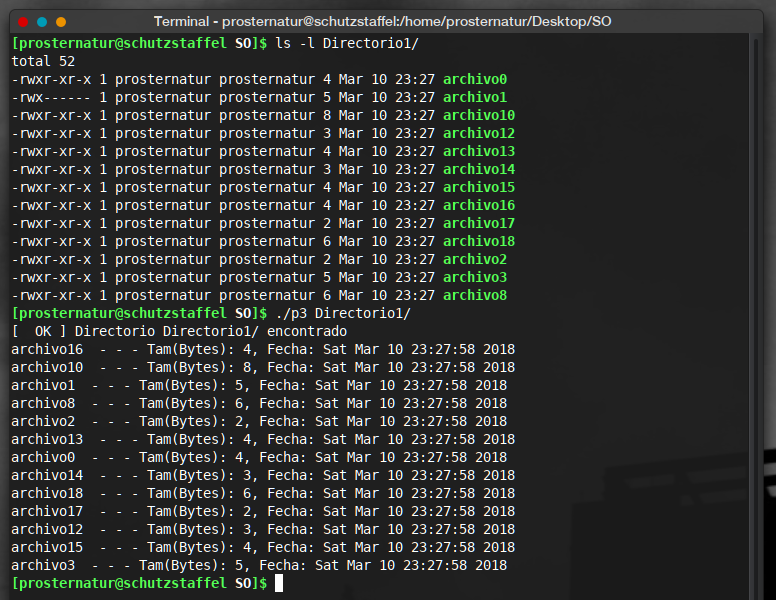
**else**

**printf("[ERROR] No existe o no se pudo acceder a %s\n", argv[1]);**

**} // ./p2 keks**

**} //stat -c '%a %n' carpetadestino/\***

Capturas de pantalla del programa que lista las propiedades de los archivos creados:

****

Captura 1: Se verificó con el comando *ls* la lista de archivos creados con el primer programa, después, con el programa en cuestión, se verificó que, en efecto, los datos mostrados fueran reales.

**11.- Una vez creados los archivos con sus contenidos por el programa del punto 8 y utilizando las llamadas al sistema revisadas para Linux que se necesitan, 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.**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <dirent.h>**

**#include <unistd.h>**

**#include <termios.h>**

**#include <fcntl.h>**

**#define BUFFER\_SZ 1024**

**int main(int argc, char \* argv[]) {**

**char archivillos[512], visualizado[512], fullarchivillos[512], contenido, \*string, \*nombre\_archivo; int i, visor, fuente, destino, bits; void \*buf = (char \* ) calloc(BUFFER\_SZ, 1);**

**archivillos[0] = '\0', fullarchivillos[0] = '\0';**

**if (argv[1] == NULL || argv[2] == NULL) {**

**printf("[ERROR] Uso: ./p4 (ruta absoluta de archivo a visualizar) (ruta absuolta destino de archivos a copiar)\n");**

**exit(-1);**

**} else {**

**DIR\* carpeta = opendir(argv[1]); //Abre la carpeta fuente**

**DIR\* carpetadestino = opendir(argv[2]); //Abre la carpeta destino**

**if (carpeta && carpetadestino) { //Verifica que ambas carpetas existan**

**printf("[ OK ] Directorio encontrado\n");**

**DIR\* d;**

**struct dirent \*carpeta;**

**d = opendir(argv[1]); //Para abrir el directorio en el que se encuentra**

**if (d) {**

**while ((carpeta = readdir(d)) != NULL) { //lista todo hasta que sea el 'fin' de la carpeta**

**if (carpeta -> d\_type == DT\_REG) { //omite cualquier cosa que no sea un archivo regular, para no mostrar carpetas despues**

**printf("%s\n", carpeta -> d\_name); //imprime todos los archivos dentro de la carpeta**

**strcat(fullarchivillos, carpeta -> d\_name); //pega a fularchivillos el nombre de la carpeta**

**strcat(fullarchivillos, ","); //y le pone una coma**

**}**

**}**

**i = strlen(fullarchivillos); //Para obtener la longitud de toda la cadena**

**fullarchivillos[i-1] = '\0'; //Quitar la ultima coma de fullarchivillos**

**closedir(d); //cierra el descriptor de la carpeta**

**chdir(argv[1]); //cambia al directorio de la primera carpeta**

**printf("\nIngrese el archivo del cual desea visualizar su contenido:\n");**

**scanf("%s", & visualizado);**

**if ((visor = open(visualizado, O\_RDONLY))>=0) //abre el archivo como solo lectura**

**{**

**printf("=========================%s========================\n", visualizado);**

**while (read(visor, & contenido, 1) == 1) //descriptor, buffer a almacenar el contenido y cantidad de bytes a leer, 1 en este caso**

**putchar(contenido); //imprime contenido byte por byte**

**printf("\n=========================================================\n");**

**close(visor); //cierra el descriptor del archivo utilizado**

**}**

**else**

**printf("[ERROR] No se pudo leer/abrir %s\n", visualizado);**

**printf("Ingrese los archivos que desea separados por comas (o enter si desea copiar todos):\n");**

**getchar();//getchar para quitar el salto de linea que puede entrar en stdin**

**fgets(archivillos,sizeof(archivillos),stdin);**

**if (archivillos[0] == '\n') //si el usuario presiona enter, entonces se utilizara fullarchivillos a continuacion, para copiar todos los archivos**

**strncpy(archivillos, fullarchivillos, 511); //reemplaza archivillos con fullarchivillos**

**else**

**strtok(archivillos, "\n"); //elimina el \n que fgets introduce**

**string = strdup(archivillos); //clona archivillos y asigna memoria dinamica para la cadena**

**while ((nombre\_archivo = strsep( & string, ",")) != NULL) //en nombre\_archivo guarda cada parte de la cadena**

**{ //hasta que encuentra un simbolo de coma, siguiendo hasta que la cadena termine**

**fuente = open(nombre\_archivo, O\_RDONLY); //abre la fuente como solo lectura**

**if (fuente != -1) //si fue posible abrir la fuente, procede**

**{**

**chdir(argv[2]); //cambia al directorio destino**

**creat(nombre\_archivo, 0755); //crea un archivo cualquiera con el mismo nombre**

**destino = open(nombre\_archivo, O\_WRONLY | O\_APPEND); //obtener descriptor del nuevo archivo**

**while ((bits = read(fuente, buf, BUFFER\_SZ)) > 0) //lee todo el archivo y guarda en bits la cantidad de bits que ha leido**

**{**

**if (write(destino, buf, bits) != bits) //escribe en el archivo destino cierta cantidad de bits, y comprueba si se escribio todo**

**printf("[ERROR] No se pudo copiar todo el buffer\n");**

**else**

**printf("[ OK ] %s copiado correctamente\n", nombre\_archivo); //solo si copia y accede correctamente al archivo**

**}**

**close(destino); close(fuente); //liberacion de descriptores**

**chdir(argv[1]); //cambiar a directorio fuente**

**} else**

**printf("[ERROR] No existe (o no se pudo leer) '%s'\n", nombre\_archivo);**

**}**

**free(buf); //liberar el buffer encargado de copiar despues de usarlo**

**return (0);**

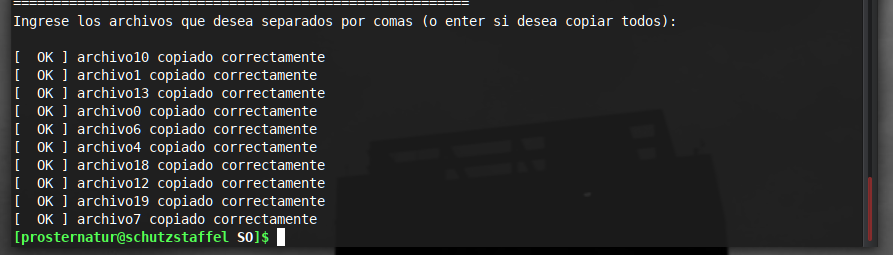
**}**

**} else**

**printf("[ERROR] Alguna de las dos carpetas no existe.\n");**

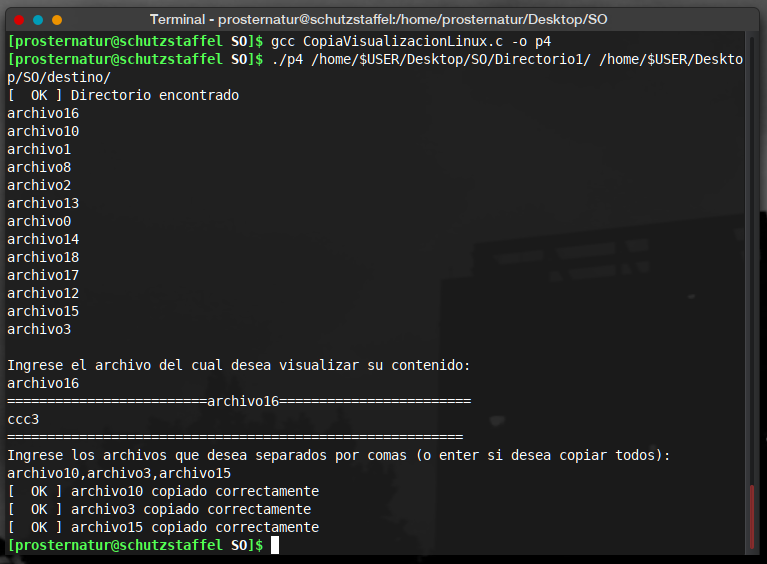
**}**

**}**

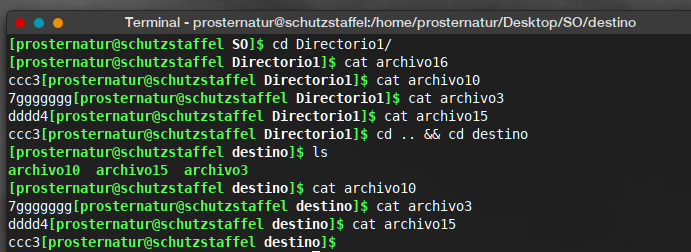
****

Captura 1: La copia de todos los archivos presionando enter es correcta.

Captura 2: Compilación y ejecución:

****

Captura 3: Verificación de que la copia y la lectura hayan sido correctas:

****

**Todo el código fuente de los programas anteriores (para Linux) se encuentra en el siguiente enlace acortado de GitHub: http://bit.do/pract2**

**12.- Desarrolle las versiones para Windows de los programas descritos en los puntos 8, 10 y 11, utilizando las llamadas al sistema revisadas para Windows que sean necesarias.**

****

**Programa 8.**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <time.h>**

**#include <sys/types.h>**

**#include <dirent.h>**

**#include<windows.h>**

**int main(int argc, char \*argv[]){**

**const char \*Arreglo[] = {"info1", "info2", "info3", "info4", "info5", "info6", "info7"};**

**if (argv[1]==NULL){ //Esto unicamente pide que un directorio despues del nombre ejecutable**

**printf("[ ERROR ] Use: ./main (ruta del directorio)\n");**

**exit(-1);**

**}else{**

**DIR\* dir = opendir(argv[1]); //Se verifica si ya existe el directorio**

**if (dir){**

**printf("[ ERROR ] Directorio ya existente\n");**

**closedir(dir);**

**exit(-1);**

**}else{**

**printf("[ OK ] Creando nuevo directorio \n");**

**CreateDirectory (argv[1], NULL);**

**printf("[ OK ] Directorio %s creado exitosamente\n",argv[1]);**

**SetCurrentDirectory(argv[1]); //Entra al directorio creado**

**int num;**

**srand(time(NULL)); //Creamos un numero aleatorio del 1 al 20**

**num = 1 + rand() % (21 - 1);**

**printf("[ INFO ] Cantidad de archivos aleatorios creados: %d \n",num);**

**int alea[num], abrir, InfoAleatoria, i, tamArreglo;**

**char nombre[15], numero[5];**

**tamArreglo = sizeof(Arreglo) / (2\*sizeof(int));**

**HANDLE hFile; //Se crea el descriptor o ayuda**

**BOOL BanderaError = FALSE;**

**for(i=1;i<=num;i++){**

**nombre[0] = '\0'; //Crearemos el nombre del primer archivo**

**strcat(nombre,"Archivo"); //Escribimos la palabra archivo**

**sprintf(numero, "%d", i); // convierte el numero a cadena para utilizar**

**strcat(nombre,numero); //Pega el numero al final del nombre**

**InfoAleatoria=rand()%tamArreglo;**

**DWORD BytesEscribir = (DWORD)strlen(Arreglo[InfoAleatoria]); //Calcula los Bytes a escribir**

**DWORD BytesEscritos = 0;**

**hFile = CreateFile(nombre, // name del archivo**

**GENERIC\_WRITE, // Abrir para escribir**

**0, // No compartir**

**NULL, // Seguridad por default**

**CREATE\_NEW, // Crear nuevo archivo**

**FILE\_ATTRIBUTE\_NORMAL, // Archivo normal**

**NULL);**

**if (hFile == INVALID\_HANDLE\_VALUE){**

**printf("[ ERROR ] Archivo no creado\n");**

**return;**

**}else{**

**BanderaError = WriteFile( hFile,**

**Arreglo[InfoAleatoria],**

**BytesEscribir,**

**&BytesEscritos,**

**NULL);**

**if (BanderaError == FALSE){**

**printf("[ ERROR ] Archivo creado pero no escrito \n");**

**}else{**

**CloseHandle(hFile);**

**}**

**}**

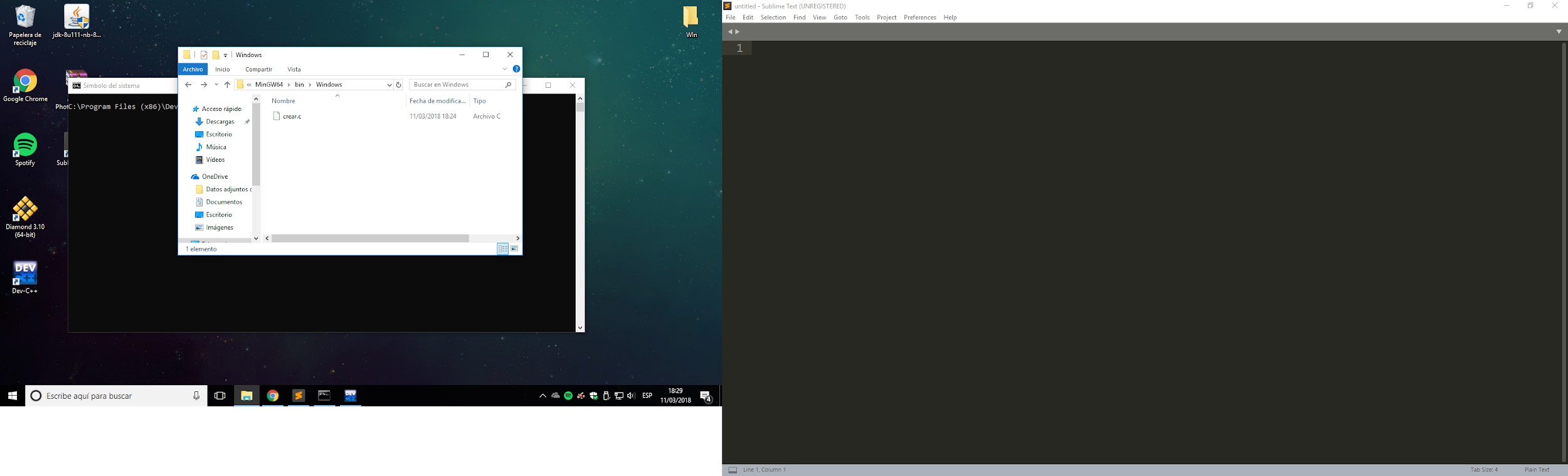
**}**

**}**

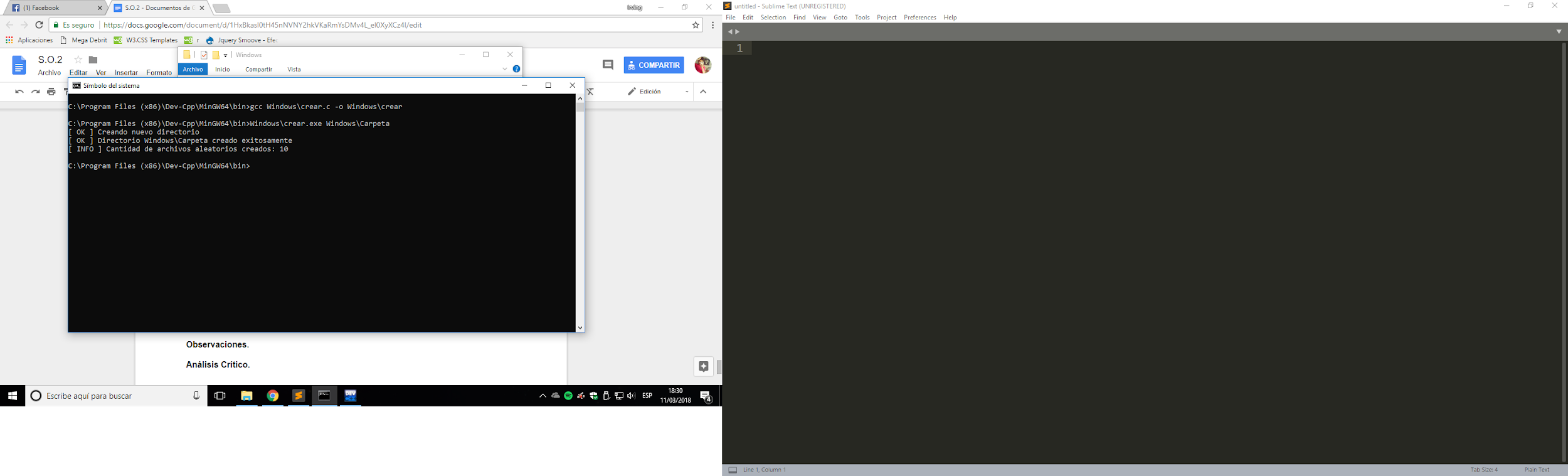
**}**

**}**

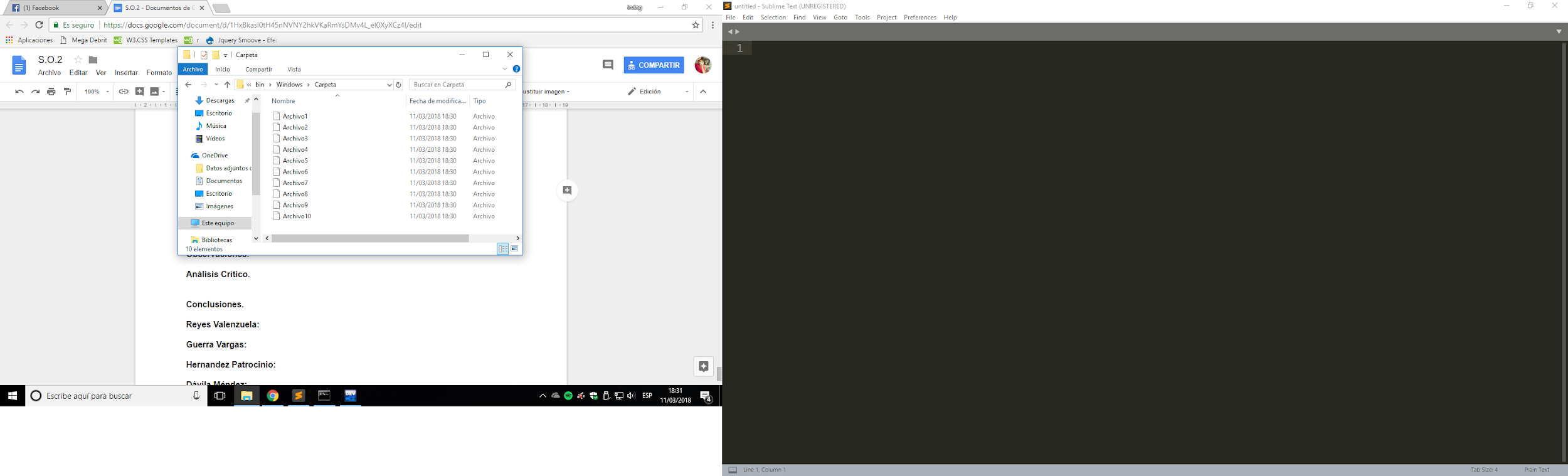
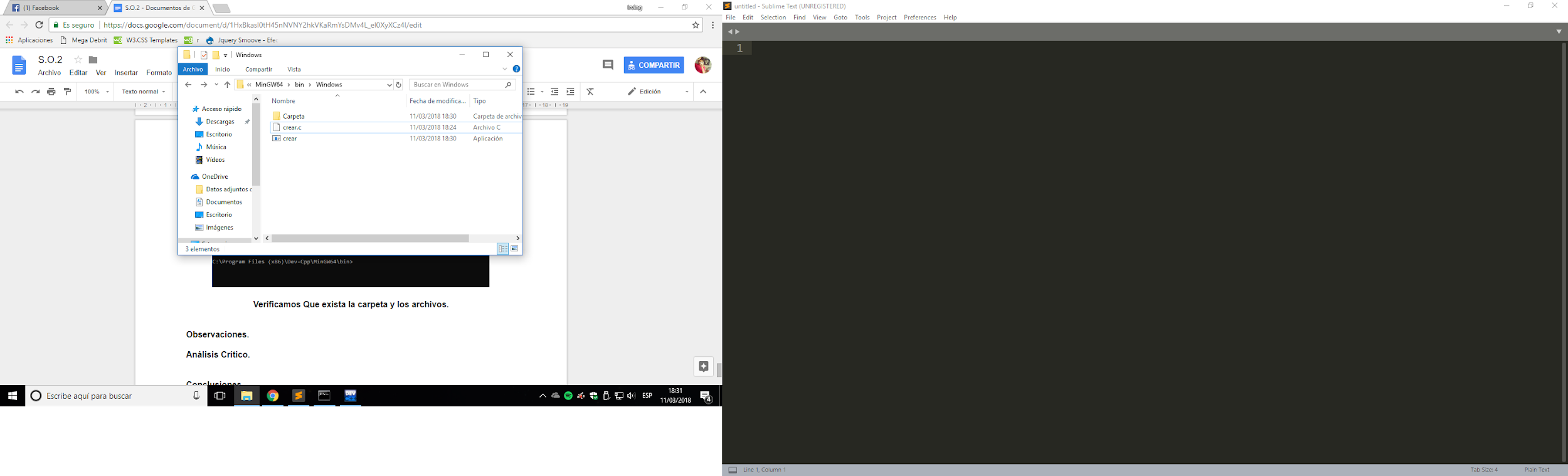
**Verificamos el directorio donde esta el .c**

****

**Compilamos, y ejecutamos, y verificamos de nuevo el directorio.**

****

**Verificamos que exista la carpeta y los archivos.**

****

**Programa 10.**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <time.h>**

**#include <sys/types.h>**

**#include <dirent.h>**

**#include<windows.h>**

**#include <unistd.h>**

**#include <sys/stat.h>**

**int main(int argc, char \*argv[]){**

**struct stat datosArchivo;**

**if (argv[1]==NULL){ //Esto unicamente pide que un directorio despues del nombre ejecutable**

**printf("[ ERROR ] Use: ./main (ruta del directorio)\n");**

**exit(-1);**

**}else{**

**DIR \*dir = opendir(argv[1]); //Se verifica si ya existe el directorio**

**if (dir){**

**printf("[ OK ] Directorio %s encontrado\n", argv[1]);**

**SetCurrentDirectory(argv[1]);**

**DIR \*d;**

**struct dirent \*dir; //la estructura dirent proporciona datos acerca de los archivos**

**d = opendir("."); //Para abrir el directorio en el que se encuentra**

**if (d){**

**while ((dir = readdir(d)) != NULL) { //lista todo hasta que sea no hayan archivos en la carpeta**

**if(dir->d\_namlen>=8){**

**stat(dir->d\_name, &datosArchivo);**

**printf("%s", dir->d\_name);**

**printf(" - - - Tam(Bytes): %d, Fecha: %s ",datosArchivo.st\_size,ctime(&datosArchivo.st\_atime));**

**}**

**}**

**}**

**}else{**

**printf("[ ERROR ] No existe o no se pudo acceder a %s\n", argv[1]);**

**closedir(dir);**

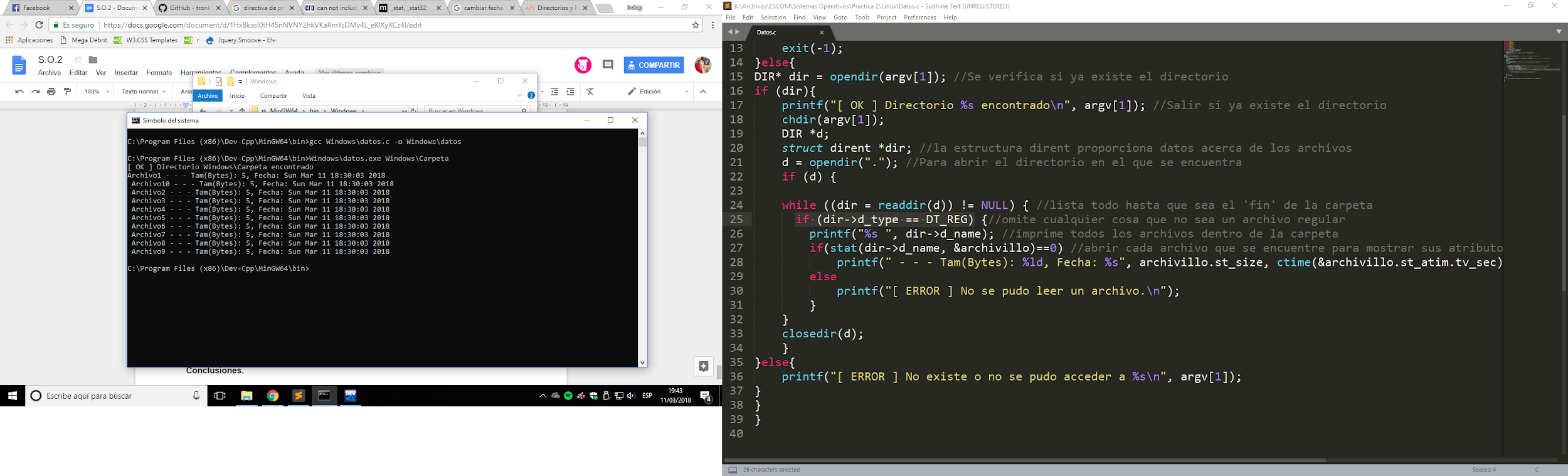
**exit(-1);**

**}**

**}**

**}**

**Compilacion y ejecucion**

****

**Programa 11.**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <dirent.h>**

**#include <unistd.h>**

**#include <string.h>**

**#define BUFFER\_SZ 1024**

**#include <windows.h>**

**int main(int argc, char \*argv[]){**

**char archivos[512], visualizado[512], fullarchivos[512], contenido[20],contenidocp[20], \*cadena, \*nombre\_archivo;**

**int i, fuente, destino, bits;**

**void \*buf = (char \* ) calloc(BUFFER\_SZ, 1);**

**archivos[0] = '\0', fullarchivos[0] = '\0';**

**if (argv[1]==NULL || argv[2] == NULL){ //Esto unicamente pide que un directorio despues del nombre ejecutable**

**printf("[ ERROR ] Use: ./main (ruta absoluta de archivo a visualizar) (ruta absuolta destino de archivos a copiar)\n");**

**exit(-1);**

**}else{**

**DIR \*carpeta = opendir(argv[1]); //Se verifica si ya existe el directorio**

**DIR \*destino = opendir(argv[2]); //Se verifica si ya existe el directorio**

**if (carpeta && destino) {**

**printf("[ OK ] Directorio encontrado\n");**

**DIR\* d;**

**DIR\* f;**

**struct dirent \*carpeta;**

**d = opendir(argv[1]);**

**if (d) {**

**while ((carpeta = readdir(d)) != NULL) {**

**if(carpeta->d\_namlen>=8) {**

**printf("%s\n", carpeta -> d\_name);**

**strcat(fullarchivos, carpeta -> d\_name);**

**strcat(fullarchivos, ","); //y le pone una coma**

**}**

**}**

**i = strlen(fullarchivos); //Para obtener la longitud de toda la cadena**

**fullarchivos[i-1] = '\0'; //Quitar la ultima coma**

**closedir(d); //cierra el descriptor de la carpeta**

**SetCurrentDirectory(argv[1]);**

**printf("\nIngrese el archivo del cual desea visualizar su contenido: ");**

**scanf("%s", &visualizado);**

**HANDLE hFile,hFile2;**

**BOOL BanderaError = FALSE;**

**BOOL BanderaError2 = FALSE;**

**DWORD BytesLeer = 20;**

**DWORD BytesEscritos = 0;**

**DWORD BytesEscribir;**

**hFile = CreateFile(visualizado, // name del archivo**

**GENERIC\_READ, // Abrir para leer**

**0, // No Compartir**

**NULL, // Seguridad por default**

**OPEN\_EXISTING, // Abre el existente**

**FILE\_ATTRIBUTE\_NORMAL, // Archivo normal**

**NULL);**

**if (hFile == INVALID\_HANDLE\_VALUE){**

**printf("[ ERROR ] Archivo no abierto\n");**

**return;**

**}else{**

**BanderaError = ReadFile( hFile,**

**contenido,**

**BytesLeer,**

**&BytesEscritos,**

**NULL);**

**if (BanderaError == FALSE){**

**printf("[ ERROR ] Archivo no leido \n");**

**}else{**

**printf("=========================%s========================\n", visualizado);**

**printf("%s",contenido);**

**printf("\n=========================================================\n");**

**CloseHandle(hFile);**

**}**

**}**

**printf("Ingrese los archivos que desea copiar, separados por comas: ");**

**getchar();**

**fgets(archivos,sizeof(archivos),stdin);**

**strtok(archivos, "\n");**

**cadena = strdup(archivos); //Clona archivos a cadena**

**nombre\_archivo = strtok (cadena,",");**

**int contador=0;**

**while (nombre\_archivo != NULL){**

**hFile = CreateFile(nombre\_archivo,**

**GENERIC\_READ,**

**0,**

**NULL,**

**OPEN\_EXISTING,**

**FILE\_ATTRIBUTE\_NORMAL,**

**NULL);**

**if (hFile == INVALID\_HANDLE\_VALUE){**

**printf("[ ERROR ] Archivo no encontrado\n");**

**return;**

**}else{**

**BanderaError = ReadFile( hFile,**

**contenidocp,**

**BytesLeer,**

**&BytesEscritos,**

**NULL);**

**if (BanderaError == FALSE){**

**printf("[ ERROR ] Archivo no leido \n");**

**}else{**

**printf("[ OK ] %s con el contendio: %s\n",nombre\_archivo,contenidocp);**

**BytesEscribir = (DWORD)strlen(contenidocp);**

**}**

**}**

**SetCurrentDirectory("..");**

**SetCurrentDirectory("..");**

**SetCurrentDirectory(argv[2]);**

**hFile2 = CreateFile(nombre\_archivo,**

**GENERIC\_WRITE,**

**0,**

**NULL,**

**CREATE\_NEW,**

**FILE\_ATTRIBUTE\_NORMAL,**

**NULL);**

**if (hFile2 == INVALID\_HANDLE\_VALUE){**

**printf("[ ERROR ] Archivo no creado\n");**

**return;**

**}else{**

**BanderaError2 = WriteFile( hFile2,**

**contenidocp,**

**BytesEscribir,**

**&BytesEscritos,**

**NULL);**

**if (BanderaError2 == FALSE){**

**printf("[ ERROR ] Archivo no escrito \n");**

**}else{**

**printf("[ OK ] %s copiado a %s\n",nombre\_archivo,argv[2]);**

**CloseHandle(hFile2);**

**}**

**}**

**nombre\_archivo = strtok (NULL, ",");**

**SetCurrentDirectory("..");**

**SetCurrentDirectory("..");**

**SetCurrentDirectory(argv[1]);**

**}**

**}**

**}else{**

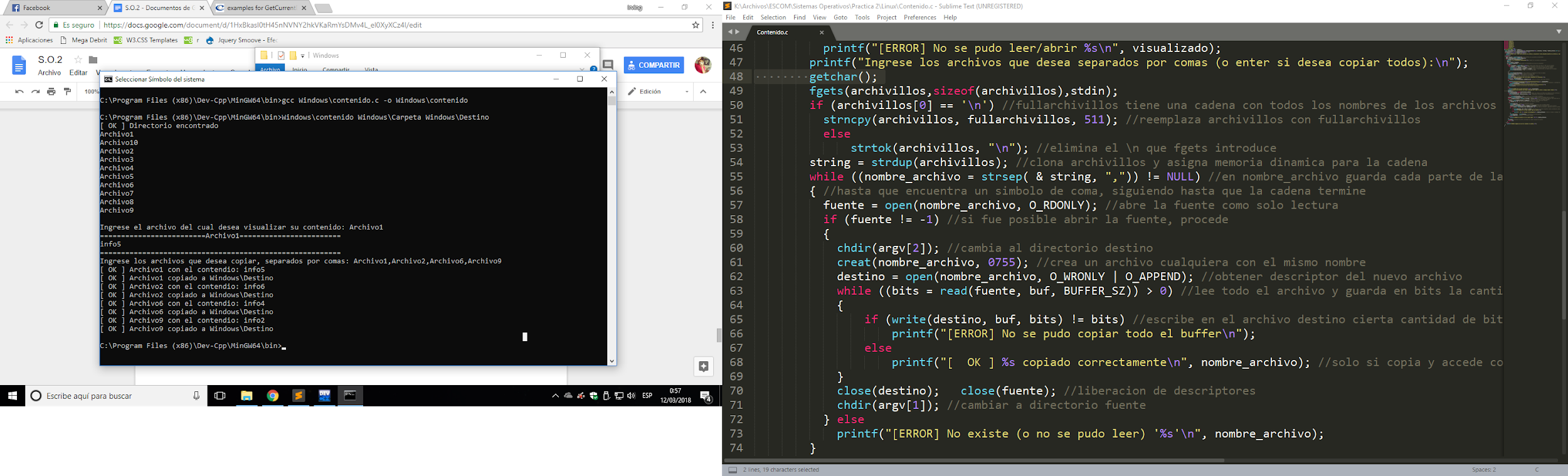
**printf("[ ERROR ] Alguna de las dos carpetas no existe.\n");**

**}**

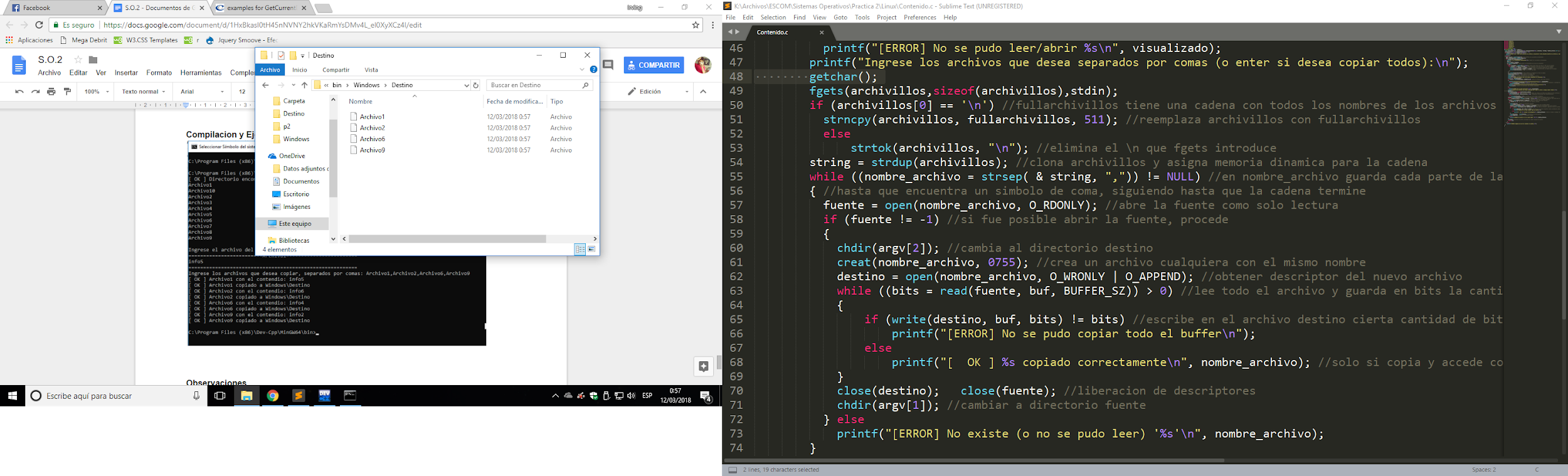
**}**

**}**

**Compilacion y Ejecucion del codigo:**

****

**Corroborando la carpeta Destino:**

****

**Observaciones:**

* Las llamadas al sistema en Windows son ligeramente más complicadas de usar.
* Ciertas llamadas al sistema de Linux son, hasta cierto punto, iguales a las que se usan en la terminal.
* La implementación de llamadas al sistema en Linux es más transparente, pues los argumentos de entrada son muy explícitos.
* No fue posible verificar el código fuente de los comandos respectivos en Windows (dir, cp, etc…) para apoyarse en la creación de los programas mostrados en este documento.
* Microsoft implementa funciones que imitan llamadas al sistema de Linux
* El manejo de directorios en Windows tiene más restricciones

**Análisis Crítico.**

Ambos sistemas operativos ofrecen llamadas al sistema que, de una u otra manera funcionan muy similarmente. Así mismo, en el caso de Linux, es posible revisar el código fuente de ciertos comandos, por ejemplo, *cp, cat, ls*, etc… lo cual puede facilitar la creación de los programas en esta práctica, no siendo así en el caso de Windows. En el caso de este último, la implementación fue un tanto más tediosa, debido a que sus respectivas llamadas al sistema implementan más argumentos que, comparadas a las de Linux, son innecesariamente más complejas. Así mismo, la falta de compatibilidad entre ambos sistemas es remarcable, pues no sólo es a nivel binaria (es decir, ejecutar algo creado con Linux en Windows), sino que, por ejemplo, los archivos creados en Word no pueden ser simplemente abiertos con algún editor de textos normal.

**Conclusiones.**

**Reyes Valenzuela:**

**Guerra Vargas:** Con esta práctica se pueden observar el comportamiento de los dos sistemas operativos, y las opciones de desarrollador que ofrecen, por una parte linux, es mas enfocado a este ámbito, y proporciona más llamadas al sistema y un fácil manejo de ellas, por otro lado, en Windows, es mas dificil o mas laborioso conseguir estas llamadas al sistema. Ahí podemos observar el enfoque de ambos sistemas operativos.

Por otro lado, el manejo es prácticamente el mismo, las funciones son muy similares, y mencionar que Windows ha tenido que “copiar” algunas llamadas al sistema que linux usa, para que los desarrolladores de windows puedan trabajar parejamente.

**Hernandez Patrocinio:**

**Dávila Méndez:** Mediante el desarrollo de esta práctica, se ha podido observar que ambos sistemas operativos ofrecen su propio conjunto de herramientas para realizar diversas acciones, siendo manipulación y creación de archivos y carpetas en esta práctica.

Así mismo, cabe destacar que las llamadas al sistema en Linux son, al parecer, más simples que aquellas utilizadas en Windows, no sólo porque necesitan menos parámetros, sino porque trabajan de manera similar a aquellas que son utilizadas en una terminal. En el caso de Windows, varios comandos no tienen su equivalente en la consola del sistema, lo que hace el uso de las llamadas al sistema más tedioso y tardado.