

INSTITUTO POLITÉCNICO NACIONAL

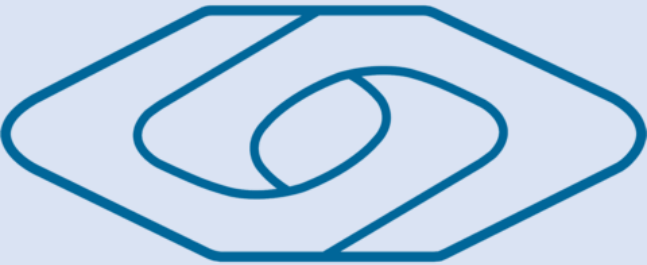
ESCUELA SUPERIOR DE COMPUTO



“Practica 3”

Sistemas Operativos

INSTITUTO POLITÉCNICO NACIONAL



ESCOM

Equipo 7

- Héctor Chávez Rodríguez
 - María Fernanda Delgado Mendoza
 - Jonathan Said Gómez Marbán
 - Luis Antonio Ramírez Fárias
-
- **Grupo:** 4CM1

I. Introducción Teórica

¿Qué es un comando Linux?

Un comando es una instrucción que le indica al sistema operativo una tarea a realizar. En algunas ocasiones hay comandos básicos que permiten crear, modificar o mover archivos y carpetas. En otras ocasiones el comando permite correr o ejecutar un programa o proceso en el S.O. Para poder ejecutar comandos se requiere de una terminal o un programa que tenga la capacidad de ejecutar comandos a través por ejemplo, de la biblioteca system.h para programas de c o de c++.

¿Qué es una terminal?

Una terminal es un programa que puede ejecutar comandos. Antes de que existieran elementos gráficos como botones o elementos de entrada de texto de forma gráfica, la terminal era la única opción para interactuar con el S.O.

Los 10 comandos Linux básicos

1. **whoami** nos indica el usuario que tiene la sesión de la computadora activa.
2. **pwd** indicará la ubicación en donde se encuentre el cursor.
3. **ls** mostrará una lista de archivos y carpetas que se encuentran en la ubicación actual.
4. **clear** limpiará la terminal del texto que se vaya juntando.
5. **history** nos indica el historial de comandos Linux que se han ejecutado desde la primera vez que se instalo el S.O.
6. **mkdir** creará una carpeta.
7. **rmdir** borra directorios vacíos.
8. **sudo** asignará los privilegios del administrador. Se utiliza cuando queremos ejecutar un comando con los máximos permisos o para crear archivos que sólo el administrador pueda modificar.
9. **cd** permite cambiar la ubicación de la terminal para ingresar a una carpeta distinta de la actual.
10. **rm** borrará archivos, por ejemplo.

La consola de Windows también existe y es una opción recomendable para que usuarios medios o profesionales realicen tareas de forma más flexible y rápida con el sistema operativo.

En sistemas Windows, la consola se denomina símbolo del sistema (*Command prompt* – línea de comandos o CMD) y es la aplicación utilizada en sistemas basados en NT (Windows XP, Windows 7, Windows 8, Windows 10, Windows Server) para ejecutar comandos MS-DOS (.exe de 16 bits) y otros como scripts con formato .bat y .sys.

La consola de Windows es la equivalente a la terminal de Linux o la que proporcionan otros sistemas operativos como macOS. La aplicación se ejecuta en modo texto y no es tan intuitiva como una interfaz gráfica de usuario, pero muestra su potencial a la hora de ejecutar tareas repetitivas, en ocasiones donde se bloquea la interfaz gráfica, para gestionar determinados componentes o acceder a cierta información del sistema que no está disponible de ninguna otra manera. Recopilamos toda la información que te hemos ido ofreciendo de esta importante herramienta.

Una vez dentro nos encontraremos con una interfaz de texto que a los usuarios que lleven tiempo en ésto les recordará poderosamente a MS-DOS, aunque no se trata del sistema operativo basado en DOS ni es una parte del sistema Windows, sino una aplicación.

Esta línea de comandos permite comunicarnos directamente con el equipo y realizar una serie de tareas. Su funcionamiento es simple a la vez que potente. Escribimos el comando y la aplicación CMD hace de intérprete para su ejecución. El uso de modificadores para cada uno de los comandos permite ejecutar centenares de combinaciones para una amplia variedad de tareas. Aunque su funcionamiento es un modo texto la consola puede ser personalizada en diseño, colores o fuentes accediendo a su propiedades mediante un clic secundario en el marco del CMD.

II. Desarrollo de la práctica

Linux

1.- Ps: El comando ps muestra por pantalla un listado de los procesos que se están ejecutando en el sistema.

Si no añadimos ningún parámetro, ps mostrará los procesos del usuario con el que estamos logeados:

```
$ ps aux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root            1  0.0  0.3  2844  1692 ?        Ss   18:13   0:01 /sbin/init
root            2  0.0  0.0      0     0 ?        S<   18:13   0:00 [kthreadd]
root            3  0.0  0.0      0     0 ?        S<   18:13   0:00 [migration/0]
root            4  0.0  0.0      0     0 ?        S<   18:13   0:00 [ksoftirqd/0]
root            5  0.0  0.0      0     0 ?        S<   18:13   0:00 [watchdog/0]
root            6  0.0  0.0      0     0 ?        S<   18:13   0:00 [migration/1]
root            7  0.0  0.0      0     0 ?        S<   18:13   0:00 [ksoftirqd/1]
root            8  0.0  0.0      0     0 ?        S<   18:13   0:00 [watchdog/1]
root            9  0.0  0.0      0     0 ?        S<   18:13   0:00 [events/0]
root           10  0.0  0.0      0     0 ?        S<   18:13   0:00 [events/1]
root           11  0.0  0.0      0     0 ?        S<   18:13   0:00 [khelper]
root           47  0.0  0.0      0     0 ?        S<   18:13   0:00 [kblockd/0]
root           48  0.0  0.0      0     0 ?        S<   18:13   0:00 [kblockd/1]
root           51  0.0  0.0      0     0 ?        S<   18:13   0:00 [kacpid]
root           52  0.0  0.0      0     0 ?        S<   18:13   0:00 [kacpi_notify]
root          128  0.0  0.0      0     0 ?        S<   18:13   0:00 [kseriod]
root          168  0.0  0.0      0     0 ?        S   18:13   0:00 [pdflush]
root          169  0.0  0.0      0     0 ?        S   18:13   0:00 [pdflush]
root          170  0.0  0.0      0     0 ?        S<   18:13   0:00 [kswapd0]
...
...
...
```

Ps-fea: Este comando sirve para gestionar el proceso de un programa o paquete que se encuentra en ejecución. Para ejecutar este comando es necesario estar en modo super usuario.

```
00 | ps -fea proceso
```

El (-fea):

-f Mostrar columnas de usuario, PID, PPID, CPU, STIME, TTY, TIME, y COMMAND

-e Seleccionar todos los procesos del usuario.

-a Lista los procesos de todos los usuarios

2.-

Comando fork()

Los procesos se crean a través haciendo una llamada fork al sistema. El nuevo proceso que se crea recibe una copia del espacio de direcciones del padre. Los dos procesos continúan su ejecución en la instrucción siguiente al fork:

```
#include
#include pid_t fork(void);
```

La creación de dos procesos totalmente idénticos no suele ser útil. Así, el valor devuelto por fork permite diferenciar el proceso padre del hijo, ya que fork devuelve 0 al hijo y el ID del hijo al padre. En caso de error devuelve -1.

La llamada fork crea procesos nuevos haciendo una copia de la imagen en la memoria del padre. El hijo hereda la mayor parte de los atributos del padre, incluyendo el entorno y los privilegios. El hijo también hereda algunos de los recursos del padre, tales como los archivos y dispositivos abiertos. Las implicaciones de la herencia son mucho más complicadas de lo que en principio pueda parecer.

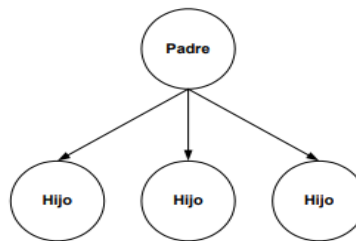


Figura 2: Árbol de procesos

Comando execv()

Las funciones `execv()`, `execvp()` y `execvpe()` proporcionan una matriz de punteros a cadenas terminadas en nulo que representan la lista de argumentos disponible para el nuevo programa. El primer argumento, por convención, debe apuntar al nombre de archivo asociado con el archivo que se está ejecutando. La matriz de punteros debe terminar con un puntero NULL.

```
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[], char *const envp[]);
```

Comando getpid()

-`getpid` devuelve el identificador de proceso del proceso actual. (Esto es usado normalmente por rutinas que generan nombres únicos de ficheros temporales.)

- `getppid` devuelve el identificador de proceso del padre del proceso actual.

```
#include <sys/types.h>
#include <unistd.h>
pid_t getpid(void);
pid_t getppid(void);
```

Comando wait()

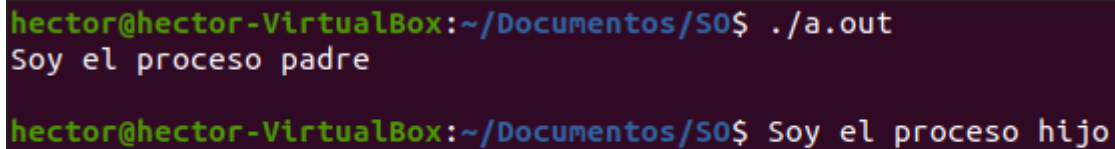
El comando wait permite al shell esperar la finalización de un proceso ejecutado en segundo plano.

```
$ wait pid1
```

3.-

Capture, compile y ejecute los dos programas de creación de un nuevo proceso por copia exacta de código que a continuación se muestra. Observe su funcionamiento y experimente con el código.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(void)
{
    int id_proc;
    id_proc=fork();
    if (id_proc == 0)
    {
        printf("Soy el proceso hijo\n");
        exit(0);
    }
    else
    {
        printf("Soy el proceso padre\n");
        exit(0);
    }
}
```



```
hector@hector-VirtualBox:~/Documentos/S0$ ./a.out
Soy el proceso padre
hector@hector-VirtualBox:~/Documentos/S0$ Soy el proceso hijo
```

```

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(void)
{
    int id_proc;
    id_proc=fork();
    if (id_proc == 0)
    {
        printf("Soy el proceso hijo\n");
    }
    else
    {
        printf("Soy el proceso padre\n");
    }
    printf("Mensaje en ambos\n");
    exit(0);
}

```

```

Soy el proceso padre
Mensaje ambos
hector@hector-VirtualBox:~/Documentos/S0$ Soy el proceso hijo
Mensaje ambos

```

4. Programe una aplicación que cree el árbol de procesos mostrado al final de este documento. Para cada uno de los procesos hijos creados se imprimirá en pantalla el pid de su padre, además se imprimirán en pantalla los pids de los hijos creados de cada proceso padre. Dibuje en papel el árbol creado usando los pid's que imprima en pantalla.

Rama 1

```

Proceso padre 2085
Soy el proceso padre 2243 del padre 2085
hector@hector-VirtualBox:~/Documentos/S0$
Proceso padre 687
Soy el proceso hijo 2244 del padre 687
Soy el proceso padre 2244 del padre 687
Soy el proceso hijo 2245 del padre 687
Soy el proceso padre 2245 del padre 687
Soy el proceso hijo 2246 del padre 687
Soy el proceso padre 2246 del padre 687
Soy el proceso hijo 2247 del padre 687
Soy el proceso padre 2247 del padre 687
Soy el proceso hijo 2248 del padre 687
Soy el proceso padre 2248 del padre 687
Soy el proceso hijo 2249 del padre 687
Soy el proceso padre 2249 del padre 687
Soy el proceso hijo 2250 del padre 687

```

Rama 2

```
Proceso padre 2085
Proceso 2
Soy el proceso padre 2292 del padre 2085
hector@hector-VirtualBox:~/Documentos/S0$
Proceso padre 687
Proceso 2
Soy el proceso hijo 2293 del padre 687
Soy el proceso padre 2293 del padre 687
Soy el proceso hijo 2294 del padre 687
Soy el proceso padre 2294 del padre 687
Soy el proceso hijo 2295 del padre 687
Soy el proceso padre 2295 del padre 687
Soy el proceso hijo 2296 del padre 687
Soy el proceso padre 2296 del padre 687
Soy el proceso hijo 2297 del padre 687
```

Rama 3

```
Proceso padre 2085
Proceso 3
Soy el proceso padre 2339 del padre 2085
hector@hector-VirtualBox:~/Documentos/S0$
Proceso padre 687
Proceso 3
Soy el proceso hijo 2340 del padre 687
Soy el proceso padre 2340 del padre 687
Soy el proceso hijo 2341 del padre 687
Soy el proceso padre 2341 del padre 687
Soy el proceso hijo 2342 del padre 687
Soy el proceso padre 2342 del padre 687
Soy el proceso hijo 2343 del padre 687
```

Rama 4

```
Proceso padre 2085
Proceso 4
Soy el proceso padre 2405 del padre 2085
hector@hector-VirtualBox:~/Documentos/S0$
Proceso padre 687
Proceso 4
Soy el proceso hijo 2406 del padre 687
Soy el proceso padre 2406 del padre 687
Soy el proceso hijo 2407 del padre 687
Soy el proceso padre 2407 del padre 687
Soy el proceso hijo 2408 del padre 687
```

Rama 5

```
Proceso padre 2085
Proceso 5
Soy el proceso padre 2447 del padre 2085
hector@hector-VirtualBox:~/Documentos/S0$
Proceso padre 687
Proceso 5
Soy el proceso hijo 2448 del padre 687
```

Código

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>

//-----
void proceso(int);
void padre();
void hijo();
void rama1(int);
void rama2(int);
void rama3(int);
void rama4(int);
void rama5(int);
void pid(int);
//-----

int main(void){

    int id_proc,pid1,pid2;

    id_proc = fork();
    pid2=getppid();
    pid(pid2);

    //-----
        rama1(id_proc);
    //-----
        rama2(id_proc);
    //-----
        rama3(id_proc);
    //-----
        rama4(id_proc);
    //-----
        rama5(id_proc);
    }
    //-----
void pid(int pid2){
    printf("\nProceso padre %d\n",getppid());
}
```



```
// getppid proceso padre
// getpid proceso hijo

void padre(){
printf("Soy el proceso padre %d del padre %d\n", getpid(), getppid());
}

void hijo(){
printf("Soy el proceso hijo %d del padre %d\n", getpid(), getppid());
}
//-----

void rama5(int id_proc){
printf("Proceso 5\n");
if(id_proc == 0){
    hijo();
    exit(0);
}
}

void rama4(int id_proc){
printf("Proceso 4\n");
if(id_proc == 0){ //rama1
hijo();
    id_proc = fork();
    if(id_proc == 0){ //rama1.1
hijo();
        id_proc = fork();
        if(id_proc == 0){ //rama1.2
hijo();
            exit(0);
        }else{
            padre();
            exit(0);
        }
    }else{
        padre();
    }
}
}

void rama3(int id_proc){
    if(id_proc == 0){ //rama1
hijo();
        id_proc = fork();
        if(id_proc == 0){ //rama1.1
```

```

        hijo();
        id_proc = fork();
        if(id_proc == 0){           //rama1.2
            hijo();
            id_proc = fork();
            if(id_proc == 0){       //rama1.3
                hijo();
                exit(0);
            }else{
                padre();
                exit(0);
            }
        }else{
            padre();
        }
    }else{
        padre();
    }
}

void rama2(int id_proc){

    if(id_proc == 0){ //rama1
        hijo();
        id_proc = fork();
        if(id_proc == 0){ //rama1.1
            hijo();
            id_proc = fork();
            if(id_proc == 0){ //rama 1.2
                hijo();
                id_proc = fork();
                if(id_proc == 0){ //rama1.3
                    hijo();
                    id_proc = fork();
                    if(id_proc == 0){ //rama1.4
                        hijo();
                        exit(0);
                    }else{
                        padre();
                        exit(0);
                    }
                }else{
                    padre();
                }
            }else{
                padre();
            }
        }else{
            padre();
        }
    }else{
        padre();
    }
}

```

```

    padre();
}

}

void ramal(int id_proc){

if(id_proc == 0){      //ramal
hijo();
    id_proc = fork();
        if(id_proc == 0){      //rama 1.1
            hijo();
                id_proc = fork();
                    if(id_proc == 0){      //ramal.2
                        hijo();
                            id_proc = fork();
                                if(id_proc == 0){      //ramal.3
                                    hijo();
                                        id_proc = fork();
                                            if(id_proc == 0){
                                                hijo();
                                                    id_proc = fork();
                                                        if(id_proc == 0){      //ramal.4
                                                            hijo();
                                                                id_proc = fork();
                                                                    if(id_proc == 0){//rama 1.5
                                                                        hijo();
                                                                            exit(0);
                                                                        }else{
                                                                            padre();
                                                                            exit(0);
                                                                        }
                                                                    }else{
                                                                        padre();
                                                                    }
                                                                }else{
                                                                    padre();
                                                                }
                                                            }else{
                                                                padre();
                                                            }
                                                        }else{
                                                            padre();
                                                        }
                                                    }else{
                                                        padre();
                                                    }
                                                }else{
                                                    padre();
                                                }
                                            }else{
                                                padre();
                                            }
                                        }else{
                                            padre();
                                        }
                                    }else{
                                        padre();
                                    }
                                }else{
                                    padre();
                                }
                            }else{
                                padre();
                            }
                        }else{
                            padre();
                        }
                    }else{
                        padre();
                    }
                }else{
                    padre();
                }
            }else{
                padre();
            }
        }else{
            padre();
        }
    }else{
        padre();
    }
}

```

5.- Programe una aplicación que cree cinco procesos (por copia exacta de código). El primer proceso se encargará de realizar la suma de dos matrices de 7x7 elementos tipo entero, el segundo proceso realizará la resta sobre esas mismas matrices, el tercer proceso realizará la multiplicación de las matrices, el cuarto proceso obtendrá las transpuestas de cada matriz. Cada uno de estos procesos escribirá un archivo con los resultados de la operación que realizó. El quinto proceso leerá los archivos de resultados y los mostrará en pantalla cada uno de ellos. Programe la misma aplicación sin la creación de procesos, es decir de forma secuencial. Obtenga los tiempos de ejecución de las aplicaciones, compare estos tiempos y dé sus observaciones.

Las funciones definidas para resolver el ejercicio (todas comentadas)

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <dirent.h>
4  #include <fcntl.h>
5  #include <errno.h>
6  #include <sys/stat.h>
7  #include <stdlib.h>
8  #include <unistd.h>
9  #include <time.h>
10 #include <unistd.h>
11 #include <time.h>
12 #define FILAS 7
13 #define COLUMNAS 7
14 #include <sys/types.h>
15 #include <unistd.h>
16 #include <time.h>
17 /*recibirDireccion regresa una cadena, esta cadena es la direccion con la
18    que trabajará cada uno de las demás funciones*/
19
20 void crearArchivo(char* direccion){
21
22     int fd1 = open(direccion, O_CREAT | O_WRONLY, 0666);
23
24     close(fd1); //Cerramos el archivo
25 }

```

/*crearArchivos no regresa nada, lo que hace es crear un numero aleatorio de archivos .txt con contenido, recibe la direccion donde se crean los archivos (esta direccion ya tiene que estar creada) y un iterador que ayudará a dar nombre a los archivos y colocar el contenido a los archivos*/

```

void escribirMatriz(int mat[7][7], char* direccion){
    //Contenido posible de los archivos
    int fd1 = open(direccion, O_WRONLY | O_APPEND, 0666); //funcion que ejecuta el comando open en la terminal, las banderas son que se crea o se escribe y lee el archivo
    int i, j;
    char*** texto = (char***)malloc(7*sizeof(char**));

    for(i = 0; i < 7; i++){
        texto[i] = (char**)malloc(7*sizeof(char*));
        for(j = 0; j < 7; j++){
            texto[i][j] = (char*)malloc(5*sizeof(char));
            sprintf(texto[i][j], "%d\t", mat[i][j]);

            write(fd1, texto[i][j], strlen(texto[i][j]));
        }
        write(fd1, "\n", 1);
    }

    close(fd1); //Cerramos el archivo
}

```

```
/*inicializarMatriz1 no regresa nada, rellena la matriz del 1 al 7 en cada renglon*/
void inicializarMatriz1(int mat1[FILAS][COLUMNAS]){
    int i, j;
    int numeros[7] = {1,2,3,4,5,6,7};
    //Se llena la matriz 1 con valores del 1 al 7
    for(i = 0; i < 7; i++){
        for(j = 0; j < 7; j++){
            mat1[i][j] = numeros[j];
        }
    }
}

/*inicializarMatriz2 no regresa nada, rellena la matriz del 7 al 1 en cada renglon*/
void inicializarMatriz2(int mat2[FILAS][COLUMNAS]){
    int i, j;
    int numeros[7] = {1,2,3,4,5,6,7};
    //Se llena la matriz 2 con valores del 7 al 1
    for(i = 0; i < 7; i++){
        int k=6;
        for(j = 0; j < 7; j++){
            mat2[i][j] = numeros[k];
            k--;
        }
    }
}

/*sumarMatrices no regresa nada, suma la matriz pasada como primer parametro, con el
segundo parametro y guarda el resultado en el tercer parametro*/
void sumarMatrices(int mat1[7][7], int mat2[7][7], int mat3[7][7]){
    int i, j;
    //Se suman los elementos de las matrices
    for(i = 0; i < 7; i++){
        for(j = 0; j < 7; j++){
            mat3[i][j] = mat1[i][j] + mat2[i][j];
        }
    }
}

/*restarMatrices no regresa nada, resta la matriz pasada como primer parametro, con el
segundo parametro y guarda el resultado en el tercer parametro*/
void restarMatrices(int mat1[7][7], int mat2[7][7], int mat3[7][7]){
    int i, j;
    //Se restan los elementos de las matrices
    for(i = 0; i < 7; i++){
        for(j = 0; j < 7; j++){
            mat3[i][j] = mat1[i][j] - mat2[i][j];
        }
    }
}
```

```
/*multiplicarMatrices no regresa nada, multiplica la matriz pasada como primer parametro, con el
segundo parametro y guarda el resultado en el tercer parametro*/
void multiplicarMatrices(int mat1[7][7], int mat2[7][7], int mat3[7][7]){
    int c,d,k, sum=0;
    //Se multiplican los elementos de las matrices
    for (c = 0; c < 7; c++){
        for (d = 0; d < 7; d++){
            for (k = 0; k < 7; k++){
                sum = sum + mat1[c][k]*mat2[k][d];
            }
            mat3[c][d] = sum;
            sum = 0;
        }
    }
}

/*imprimirMatriz no regresa nada, pero imprime todos los renglones y columnas*/
void imprimirMatriz(int mat[7][7]){
    int i, j;
    printf("\n\nMatriz: ");
    //Se imprimen los elementos de la matriz
    for(i = 0; i < 7; i++){
        printf("\n");
        for(j = 0; j < 7; j++){
            printf("%d ", mat[i][j]);
        }
    }
}

//transpuesta, recibe dos matrices, lo que hace es la transpuesta del primer parametro lo coloca en el segundo
void transpuesta(int mat1[7][7], int mat2[7][7])
{
    int i, j;
    for (i = 0; i < 7; i++)
        for (j = 0; j < 7; j++)
            mat2[i][j] = mat1[j][i];
}
```

Después, se pide que se haga de manera concurrente por copia exacta de código (función fork) y de manera secuencial. Presentamos en primer lugar la que es de manera concurrente:

```
int main(){
    int mat1[7][7], mat2[7][7], matSum[7][7], matRes[7][7], matMul[7][7], matTrans[7][7];

    inicializarMatriz1(mat1);
    imprimirMatriz(mat1);

    inicializarMatriz2(mat2);
    imprimirMatriz(mat2);

    if(fork() == 0){
        sumarMatrices(mat1,mat2,matSum);
        imprimirMatriz(matSum);
        crearArchivo("Documents/sumaDeMatrices.txt");
        escribirMatriz(matSum, "Documents/sumaDeMatrices.txt");
        exit(0);
    }

    if(fork() == 0){
        restarMatrices(mat1,mat2,matRes);
        imprimirMatriz(matRes);
        crearArchivo("Documents/restaDeMatrices.txt");
        escribirMatriz(matRes, "Documents/restaDeMatrices.txt" );
        exit(0);
    }

    if(fork() == 0){
        multiplicarMatrices(mat1,mat2,matMul);
        imprimirMatriz(matMul);
        crearArchivo("Documents/multDeMatrices.txt");
        escribirMatriz(matMul, "Documents/multDeMatrices.txt" );
        exit(0);
    }

    if(fork() == 0){
        transpuesta(mat1,matTrans);
        imprimirMatriz(matTrans);
        crearArchivo("Documents/transDeMatrices.txt");
        escribirMatriz(matTrans, "Documents/transDeMatrices.txt" );
        exit(0);
    }

    return 0;
}
```

Salidas de pantalla y archivos creados con su contenido:

```

7 6 5 4 3 2 1
Matriz:
-6 -4 -2 0 2 4 6
-6 -4 -2 0 2 4 6
-6 -4 -2 0 2 4 6
-6 -4 -2 0 2 4 6
-6 -4 -2 0 2 4 6
7 6 5 4 3 2 1
Matriz:
196 168 140 112 84 56 28
196 168 140 112 84 56 28
196 168 140 112 84 56 28
-6 -4 -2 0 2 4 6
-6 -4 -2 0 2 4 6
196 168 140 112 84 56 28
196 168 140 112 84 56 28
jona@ubuntu:~$ 196 168 140 112 84 56 28
196 168 140 112 84 56 28 -6 -4 -2 0 2 4 6 8 8 8 8 8 8
8 8 8 8 8 8
7 6 5 4 3 2 1
Matriz:
1 1 1 1 1 1 1
2 2 2 2 2 2 2
3 3 3 3 3 3 3
4 4 4 4 4 4 4
5 5 5 5 5 5 5
6 6 6 6 6 6 6
7 7 7 7 7 7 8 8 8 8 8 8 ^C
jona@ubuntu:~$ gcc Matrices.c -o Matrices
jona@ubuntu:~$

```

Ejecutando el archivo, no se tardó mas de 2 seg.

```

multDeMatrices.txt  restaDeMatrices.txt  sumaDeMatrices.txt  transDeMatrices.txt
jona@ubuntu: ~
7 6 5 4 3 2 1
Matriz:
jona@ubuntu:~$ 1 1 1 1 1 1 1
2 2 2 2 2 2 2
3 3 3 3 3 3 3
4 4 4 4 4 4 4
196 168 140 112 84 56 28
5 5 5 5 5 5 5
196 168 140 112 84 56 28
6 6 6 6 6 6 6
196 168 140 112 84 56 28
196 168 140 112 84 56 28
7 7 7 7 7 7 196 168 140 112 84 56 28 7 6 5 4 3 2 1
Matriz:
8 8 8 8 8 8 8
8 8 8 8 8 8 8
8 8 8 8 8 8 8
8 8 8 8 8 8 8
8 8 8 8 8 8 8
8 8 8 8 8 8 8
8 8 8 8 8 8 7 6 5 4 3 2 1
Matriz:
-6 -4 -2 0 2 4 6
-6 -4 -2 0 2 4 6
-6 -4 -2 0 2 4 6
-6 -4 -2 0 2 4 6
-6 -4 -2 0 2 4 6
-6 -4 -2 0 2 4 6
-6 -4 -2 0 2 4 6

```

Contenido de archivos creados:

sumaDeMatrices.txt ~/Documents							
1	8	8	8	8	8	8	8
2	8	8	8	8	8	8	8
3	8	8	8	8	8	8	8
4	8	8	8	8	8	8	8
5	8	8	8	8	8	8	8
6	8	8	8	8	8	8	8
7	8	8	8	8	8	8	8

restaDeMatrices.txt ~/Documents							
sumaDeMatrices.txt				restaDeMatrices.txt			
1	-6	-4	-2	0	2	4	6
2	-6	-4	-2	0	2	4	6
3	-6	-4	-2	0	2	4	6
4	-6	-4	-2	0	2	4	6
5	-6	-4	-2	0	2	4	6
6	-6	-4	-2	0	2	4	6
7	-6	-4	-2	0	2	4	6

multDeMatrices.txt ~/Documents							
multDeMatrices.txt							
1	196	168	140	112	84	56	28
2	196	168	140	112	84	56	28
3	196	168	140	112	84	56	28
4	196	168	140	112	84	56	28
5	196	168	140	112	84	56	28
6	196	168	140	112	84	56	28
7	196	168	140	112	84	56	28

transDeMatrices.txt ~/Documents							
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7

6.- Capture, compile y ejecute el siguiente programa de creación de un nuevo proceso con sustitución de un nuevo código, así como el programa que será el nuevo código para ejecutar. Observe su funcionamiento y experimente con el código

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>


int main(void){
    pid_t pid;
    char*argv[3];
    argv[0]="/home/hector/Documentos/hola";
    argv[1]="Desde el hijo";
    argv[2]=NULL;

    if((pid=fork())== -1)
        printf("Error al crear el proceso hijo");
    if(pid == 0){
        printf("Soy el hijo ejecutado: %s\n", argv[0]);
        execv(argv[0],argv);
    }else{
        wait(0);
        printf("Soy el padre\n");
        exit(0);
    }
}

#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int main(int argc, char*argv){
    char mensaje[100];
    strcpy(mensaje, "Hola mundo");
    strcat(mensaje,argv[1]);
    printf("%s\n", mensaje);
    exit(0);
}
```

```
hector@hector-VirtualBox:~/Documentos$ ./a.out
Soy el hijo ejecutado: /home/hector/Documentos/hola
Hola mundo Desde el hijo
Soy el padre
hector@hector-VirtualBox:~/Documentos$
```

Ahora lo ejecutamos de manera secuencial:



The image shows a code editor window with four tabs: 'Start here', 'Matrices.c', 'Open.c', and 'MatricesSecuenciales.c'. The 'MatricesSecuenciales.c' tab is active, displaying a C program. The code is numbered from 148 to 184. It defines a `main` function that performs several matrix operations: initialization, printing, summing, subtracting, multiplying, and transposing, with each result being saved to a text file in the 'Documents' directory. The code uses arrays of size 7x7 and includes function calls for each operation.

```
148     }
149 }
150 }
151
152 int main(){
153     int mat1[7][7], mat2[7][7], matSum[7][7], matRes[7][7], matMul[7][7], matTrans[7][7];
154
155     inicializarMatriz1(mat1);
156     imprimirMatriz(mat1);
157
158     inicializarMatriz2(mat2);
159     imprimirMatriz(mat2);
160
161     sumarMatrices(mat1,mat2,matSum);
162     imprimirMatriz(matSum);
163     crearArchivo("Documents/sumaDeMatrices.txt");
164     escribirMatriz(matSum, "Documents/sumaDeMatrices.txt");
165
166     restarMatrices(mat1,mat2,matRes);
167     imprimirMatriz(matRes);
168     crearArchivo("Documents/restaDeMatrices.txt");
169     escribirMatriz(matRes, "Documents/restaDeMatrices.txt" );
170
171     multiplicarMatrices(mat1,mat2,matMul);
172     imprimirMatriz(matMul);
173     crearArchivo("Documents/multDeMatrices.txt");
174     escribirMatriz(matMul, "Documents/multDeMatrices.txt" );
175
176     transpuesta(mat1,matTrans);
177     imprimirMatriz(matTrans);
178     crearArchivo("Documents/transDeMatrices.txt");
179     escribirMatriz(matTrans, "Documents/transDeMatrices.txt" );
180
181
182     return 0;
183 }
184
```

```
jona@ubuntu:~$ gcc MatricesSecuenciales.c -o MatricesSecuenciales
jona@ubuntu:~$ ./MatricesSecuenciales

Matriz:
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7

Matriz:
7 6 5 4 3 2 1
7 6 5 4 3 2 1
7 6 5 4 3 2 1
7 6 5 4 3 2 1
7 6 5 4 3 2 1
7 6 5 4 3 2 1
7 6 5 4 3 2 1

Matriz:
8 8 8 8 8 8 8
8 8 8 8 8 8 8
8 8 8 8 8 8 8
8 8 8 8 8 8 8
8 8 8 8 8 8 8
8 8 8 8 8 8 8
```

```
Matriz:
-6 -4 -2 0 2 4 6
-6 -4 -2 0 2 4 6
-6 -4 -2 0 2 4 6
-6 -4 -2 0 2 4 6
-6 -4 -2 0 2 4 6
-6 -4 -2 0 2 4 6
-6 -4 -2 0 2 4 6

Matriz:
196 168 140 112 84 56 28
196 168 140 112 84 56 28
196 168 140 112 84 56 28
196 168 140 112 84 56 28
196 168 140 112 84 56 28
196 168 140 112 84 56 28
196 168 140 112 84 56 28

Matriz:
1 1 1 1 1 1 1
2 2 2 2 2 2 2
3 3 3 3 3 3 3
4 4 4 4 4 4 4
5 5 5 5 5 5 5
6 6 6 6 6 6 6
7 7 7 7 7 7 7 jona@ubuntu:~$
```

Al ejecutarse notamos que se ha tardado unos 5 seg, un poco mas de tiempo que haciendo uso de fork.

Y a generado el mismo contenido en los archivos creados:

Windows

Se ejecuta el siguiente programa junto con el proceso hijo:

```
#include <windows.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
    STARTUPINFO si;           /* Estructura de información inicial para Windows */
    PROCESS_INFORMATION pi;    /* Estructura de información del adm. de procesos */
    int i;
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));
    if(argc!=2)
    {
        printf("Usar: %s Nombre_programa_hijo\n", argv[0]);
        return;
    }

    // Creación proceso hijo
    if(!CreateProcess(NULL, argv[1], NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
    {
        printf( "Fallo al invocar CreateProcess (%d)\n", GetLastError() );
        return;
    }

    // Proceso padre
    printf("Soy el padre\n");
    WaitForSingleObject(pi.hProcess, INFINITE);

    // Terminación controlada del proceso e hilo asociado de ejecución
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}

#include <windows.h>
#include <stdio.h>

int main(void)
{
    printf("Soy el hijo\n");
    exit(0);
}

#include <windows.h>
#include <stdio.h>

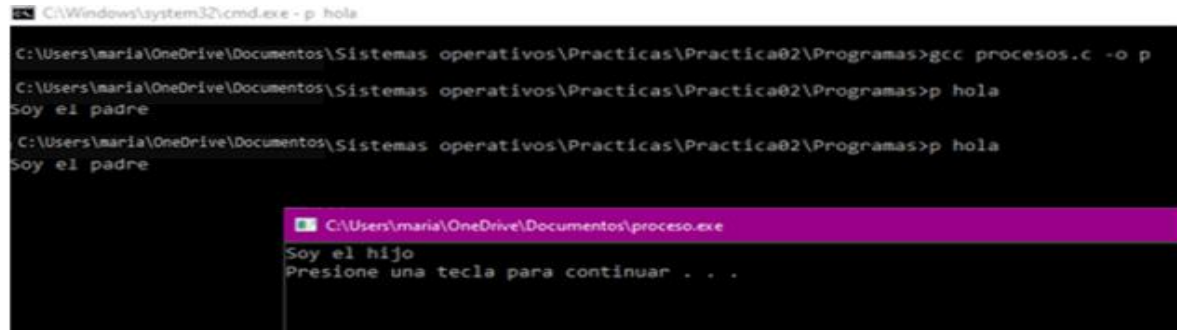
int main(void)
{
    printf("Soy el hijo\n");
    exit(0);
}

#include <windows.h>
#include <stdio.h>

int main(void)
{
    printf("Soy el hijo\n");
    exit(0);
}
```

Aquí podemos observar la ejecución del proceso padre y la ejecución del proceso hijo, sin embargo, el proceso hijo se ejecuta en una ventana nueva de la línea de comandos, esto es debido a que añadimos un modificador a la línea de `CreateProcess` como se muestra a

continuación:



```

C:\Windows\system32\cmd.exe -p hola

C:\Users\maria\OneDrive\Documentos\Sistemas operativos\Practicas\Practica02\Programas>gcc procesos.c -o p
C:\Users\maria\OneDrive\Documentos\Sistemas operativos\Practicas\Practica02\Programas>p hola
Soy el padre

C:\Users\maria\OneDrive\Documentos\Sistemas operativos\Practicas\Practica02\Programas>p hola
Soy el padre

C:\Users\maria\OneDrive\Documentos\proceso.exe
Soy el hijo
Presione una tecla para continuar . . .

#include <windows.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
    STARTUPINFO si; /* Estructura de información inicial para Windows */
    PROCESS_INFORMATION pi; /* Estructura de información del adm. de procesos */
    int i;
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));
    if(argc!=2)
    {
        printf("Usar: %s Nombre_programa_hijo\n", argv[0]);
        return 1;
    }
    // Creación proceso hijo
    if(!CreateProcess(NULL, argv[1], NULL, NULL, FALSE, CREATE_NEW_CONSOLE, NULL, NULL, &si, &pi))
    {
        printf( "Fallo al invocar CreateProcess (%d)\n", GetLastError() );
        return -1;
    }
    // Proceso padre
    printf("Soy el padre\n");
    WaitForSingleObject(pi.hProcess, INFINITE);
    // Terminación controlada del proceso e hilo asociado de ejecución
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}

```

Como podemos observar, en el argumento de la función `CreateProcess`, se añadió el modificador `CREATE_NEW_CONSOLE`, esto nos permite abrir el proceso creado en una nueva consola y cuando el valor está en 0 (que es como estaba originalmente) el proceso se abre en la misma consola que el proceso padre.

Compare y reporte tanto las diferencias como similitudes que encuentra con respecto a la creación de procesos por sustitución de código en Linux.

Una de las diferencias más notables, es la forma en la que se crean procesos, en Linux los comandos son más simples y nos proporcionan mayor información sobre el proceso, es decir, está mejor definido el manejo de procesos, a diferencia de Windows, es verdad que tenemos muchas opciones en la creación de procesos, pero no nos proporciona mucha información sobre el árbol de procesos, es decir, sobre quién es el padre y quien es el hijo, solo lo sabemos por la forma en que lo creamos pero los comandos tal cual no nos proporcionan dicha información.

Programa una aplicación que cree un proceso hijo a partir de un proceso padre, el hijo creado a su vez creará 5 procesos hijos más. A su vez cada uno de los cinco procesos creará 3 procesos más. Cada uno de los procesos creados imprimirá en pantalla su identificador.

```
#include <windows.h>
#include <stdio.h>
```

```
int main(void)
{
    printf("Soy el hijo\n");
    exit(0);
}
```

```
1  #include <windows.h>
2  #include <stdio.h>
3  int main(int argc, char *argv[])
4  {
5      STARTUPINFO si; /* Estructura de información inicial para Windows */
6      PROCESS_INFORMATION pi; /* Estructura de información del ade. de procesos */
7      int i, j;
8
9      /*Se llenan los bloques de memoria con ceros*/
10     ZeroMemory(&si, sizeof(si));
11     si.cb = sizeof(si);
12     ZeroMemory(&pi, sizeof(pi));
13     /*Verifica los parametros que se pasan en la línea d comandos*/
14     if(argc!=2)
15     {
16         printf("Usar: %s Nombre_programa_hijo\n", argv[0]);
17         return 1;
18     }
19     // Creación proceso hijo
20     if(!CreateProcess(NULL, argv[1], NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
21     {
22         printf("Fallo al invocar CreateProcess (%d)\n", GetLastError());
23         return -1;
24     }
25     else
26     {
27         // Aquí se debería implementar la lógica para crear 5 hijos y cada uno de ellos 3 más.
28         // Por ejemplo, usando fork() o _spawn() en Windows.
29     }
30 }
```



```

27 //Creacion de 5 hijos para el hijo anterior
28 for(i=0;i<5;i++)
29 {
30     if(!CreateProcess(NULL, argv[1], NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
31     {
32         printf( "Fallo al invocar CreateProcess (%d)\n", GetLastError() );
33         return -1;
34     }
35     else
36     {
37         //Creacion de 3 hijos para cada uno de los 5 hijos anteriores
38         for(j=0;j<3;j++)
39         {
40             if(!CreateProcess(NULL, argv[1], NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
41             {
42                 printf( "Fallo al invocar CreateProcess (%d)\n", GetLastError());
43                 return -1;
44             }
45         }
46     }
47 }
48
49 // Proceso padre
50 printf("PROCESO PADRE: %d\n", GetCurrentProcessId());
51 WaitForSingleObject(pi.hProcess, INFINITE);
52 // Terminación controlada del proceso e hilo asociado de ejecución
53 CloseHandle(pi.hProcess);
54 CloseHandle(pi.hThread);
55 }

```

Este código muestra la creación de 1 proceso hijo, 5 procesos para dicho proceso hijo y otros 3 procesos para cada uno de los 5 procesos anteriores. En este obtenemos la ejecución de sus respectivos IDs a través de la función `GetCurrentProcessId()`. Y en el proceso hijo que creamos, también ejecutamos la función `GetCurrentProcessId()` como se muestra a continuación:

```

1 #include <windows.h>
2 #include <stdio.h>
3 int main(void)
4 {
5     printf("ESTE ES UN PROCESO CON ID: %d\n", GetCurrentProcessId());
6     Sleep(1000);
7     return 0;
8 }

```

En total, obtenemos ejecutar 24 procesos y sus respectivos ID:

```

C:\Users\maria\OneDrive\Documentos\Sistemas operativos\Practicas\Practica02\Programas>gcc proceso.c -o p
C:\Users\CRISTIAN\Documents\Sistemas operativos\Practicas\Practica02\Programas>p hijo
ESTE ES UN PROCESO CON ID: 296780
ESTE ES UN PROCESO CON ID: 138356
ESTE ES UN PROCESO CON ID: 236940
ESTE ES UN PROCESO CON ID: 277080
ESTE ES UN PROCESO CON ID: 321592
ESTE ES UN PROCESO CON ID: 309928
ESTE ES UN PROCESO CON ID: 72828
ESTE ES UN PROCESO CON ID: 309932
ESTE ES UN PROCESO CON ID: 263548
ESTE ES UN PROCESO CON ID: 221788
ESTE ES UN PROCESO CON ID: 335384
ESTE ES UN PROCESO CON ID: 309952
ESTE ES UN PROCESO CON ID: 236984
ESTE ES UN PROCESO CON ID: 263592
ESTE ES UN PROCESO CON ID: 221544
ESTE ES UN PROCESO CON ID: 79056
ESTE ES UN PROCESO CON ID: 62276
ESTE ES UN PROCESO CON ID: 54648
PROCESO PADRE: 224192
ESTE ES UN PROCESO CON ID: 335412
ESTE ES UN PROCESO CON ID: 309948
ESTE ES UN PROCESO CON ID: 221728

```

Como se observa, nos genera un proceso distinto, sin embargo, es difícil distinguir la estructura de quién es el hijo de quien, en diferencia, si hubiéramos usado Linux, no tendríamos ese problema, pues tenemos identificadores para procesos padres e hijos, cosa que en Windows no.

8. Programe las aplicaciones desarrolladas en el punto 5 de la sección de Linux (tanto la de procesos como la secuencial) utilizando esta vez la creación de procesos en Windows

- 1) Después de analizar el problema y de entender lo que nos pide, comenzamos a escribir las funciones a utilizar y definimos el total de filas y columnas.

```

#include <stdio.h>
#include <stdlib.h>
#define filas 7
#define columnas 7
// Predefinición de funciones
void SoliciteArray(int **columnas);
void VisualArray(int **columnas);
void Sum(int **columnas, int **columnas);
void Res(int **columnas, int **columnas);
void Tras(int **columnas);
void Mult(int **columnas, int **columnas);
void menu(int);

```

- 2) En el main tendremos solo la función del menú

```

14 int main()
15 {
16     int num;
17     //Menu
18     menu(num);
19     return 0;
20 }

```

- 3) A continuación, escribimos el cuerpo de las funciones escritas anterior mente en la cabecera del código.

```

21 //Menu
22 void menu (int num)
23 {
24
25     int n, opcion;
26     int MatrizA[filas][columnas];
27     int MatrizB[filas][columnas];
28
29     // Peticion de datos para la matriz a
30     printf("\nSolicitando datos para la primera matriz\n");
31     SoliciteArray(MatrizA);
32     // Peticion de datos para la matriz b
33     printf("Solicitando datos para la segunda matriz\n");
34     SoliciteArray(MatrizB);
35     printf("\n Estos son las datos ingresados\n");
36     // Visualizar los datos de la matriz a
37     VisualArray(MatrizA);
38     // Visualizar los datos de la matriz b
39     VisualArray(MatrizB);
40
41     do
42     {
43         printf( "\n    1. Calcular la suma de matrices.");
44         printf( "\n    2. Calcular la resta de matrices");
45         printf( "\n    3. Calcular la traspuesta");
46         printf( "\n    4. Calcular la multiplicacion de matrices");
47         printf( "\n    5. Salir." );
48         printf( "\n\n    Introduzca opci%cn deseada ");
49
50         scanf( "%d", &opcion );
51
52     } while (opcion != 5);
53 }
54
55 // Funcion para solicitar los datos del array
56 void SoliciteArray(int Matriz[][columnas])
57 {
58     int fila;
59     int columna;
60     printf("\n");
61     for(fila=0;fila<filas;fila++) {
62         for(columna=0;columna<columnas;columna++) {
63             printf("Posicion %d - %d :", fila, columna);
64             scanf(" %d",&Matriz[fila][columna]);
65         }
66     }
67     printf("\n");
68 }
69
70 // Funcion para visualizar los datos de la matriz
71 void VisualArray(int Matriz[][columnas]) {
72     int fila;
73     int columna;
74     printf("\n");
75     for(fila=0;fila<filas;fila++) {
76         for(columna=0;columna<columnas;columna++) {
77             printf("%5d",Matriz[fila][columna]);
78         }
79         printf("\n");
80     }
81 }
82
83 // Funcion para sumar matrices
84 void Sum(int Matriz1[][columnas],int Matriz2[][columnas]) {
85     int fila;
86     int columna;
87     int resultado[filas][columnas];
88     for(fila=0;fila<filas;fila++) {
89         for(columna=0;columna<columnas;columna++) {
90             resultado[fila][columna]=Matriz1[fila][columna]+Matriz2[fila][columna];
91         }
92     }
93 }
94
95 // Funcion para restar matrices
96 void Resta(int Matriz1[][columnas],int Matriz2[][columnas]) {
97     int fila;
98     int columna;
99     int resultado[filas][columnas];
100     for(fila=0;fila<filas;fila++) {
101         for(columna=0;columna<columnas;columna++) {
102             resultado[fila][columna]=Matriz1[fila][columna]-Matriz2[fila][columna];
103         }
104     }
105 }
106
107 // Funcion para traspuesta de matrices
108 void Traspuesta(int Matriz[][columnas]) {
109     int fila;
110     int columna;
111     int resultado[columnas][filas];
112     for(fila=0;fila<filas;fila++) {
113         for(columna=0;columna<columnas;columna++) {
114             resultado[columna][fila]=Matriz[fila][columna];
115         }
116     }
117 }
118
119 // Funcion para multiplicar matrices
120 void Multiplicar(int Matriz1[][columnas],int Matriz2[][columnas]) {
121     int fila;
122     int columna;
123     int resultado[filas][columnas];
124     for(fila=0;fila<filas;fila++) {
125         for(columna=0;columna<columnas;columna++) {
126             resultado[fila][columna]=0;
127             for(int k=0;k<columnas;k++) {
128                 resultado[fila][columna]+=Matriz1[fila][k]*Matriz2[k][columna];
129             }
130         }
131     }
132 }

```

```

129 // Funcion para restar matrices
130 void Res(int Matriz1[][columnas],int Matriz2[][columnas]) {
131     int fila;
132     int columna;
133     int resultado[filas][columnas];
134     for(fila=0;fila<filas;fila++) {
135         for(columna=0;columna<columnas;columna++) {
136             resultado[fila][columna]=Matriz1[fila][columna]-Matriz2[fila][columna];
137         }
138     }
139     // Visualizar matriz de resultado
140     VisualArray(resultado);
141 }
142 void Tras(int Matriz[][columnas]) {
143     int fila;
144     int columna;
145     printf("\nVisualizar la matriz normal\n");
146     VisualArray(Matriz);
147     printf("\nVisualizar la traspuesta de la matriz\n");
148     for(fila=0;fila<columnas;fila++) {
149         for(columna=0;columna<filas;columna++) {
150             printf("%5d",Matriz[columna][fila]);
151         }
152         printf("\n");
153     }
154 }
155 //Funcion para multiplicar matrices
156 void Mult(int Matriz1[][columnas],int Matriz2[][columnas])
157 {
158     int fila;
159     int columna;
160     int resultado[filas][columnas];
161     int suma;
162     for (fila= 0; fila< filas; fila++) { //se itera através de cada fila de matriz1
163         for (columna = 0; columna < columnas; columna++) { //se itera através de cada columna de matriz2
164             suma = 0; //es donde se almacenará el valor final
165             for (columna = 0; columna < columnas; columna++) {
166                 suma += Matriz1[fila][columna] * Matriz2[fila][columna]; //se acumula en suma
167             }
168             resultado[filas][columnas] = suma;
169         }
170     }

```

4) Solicitamos los datos de la primera matriz

Solicitando datos para la primera matriz

```

Posicion 0 - 0 :1
Posicion 0 - 1 :2
Posicion 0 - 2 :3
Posicion 0 - 3 :4
Posicion 0 - 4 :5
Posicion 0 - 5 :6
Posicion 0 - 6 :7
Posicion 1 - 0 :1
Posicion 1 - 1 :2
Posicion 1 - 2 :3
Posicion 1 - 3 :4
Posicion 1 - 4 :5
Posicion 1 - 5 :6
Posicion 1 - 6 :7

```

- 5) Solicitamos los datos de la segunda matriz

```
Solicitando datos para la segunda matriz
Posicion 0 - 0 :1
Posicion 0 - 1 :2
Posicion 0 - 2 :3
Posicion 0 - 3 :4
Posicion 0 - 4 :5
Posicion 0 - 5 :6
Posicion 0 - 6 :7
Posicion 1 - 0 :1
Posicion 1 - 1 :2
Posicion 1 - 2 :3
Posicion 1 - 3 :4
Posicion 1 - 4 :5
Posicion 1 - 5 :6
Posicion 1 - 6 :7
```

- 6) Se muestran los datos ingresados

```
Estos son las datos ingresados

1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7

1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
```

- 7) Se muestra las operaciones que se pueden realizar

```
1. Calcular la suma de matrices.
2. Calcular la resta de matrices
3. Calcular la traspuesta
4. Calcular la multiplicacion de matrices
5. Salir.

Introduzca opción deseada 1
```

- 8) Introducimos la opción 1 para realizar la suma de matrices

La suma de las matrices es la siguiente:

2	4	6	8	10	12	14
2	4	6	8	10	12	14
2	4	6	8	10	12	14
2	4	6	8	10	12	14
2	4	6	8	10	12	14
2	4	6	8	10	12	14
2	4	6	8	10	12	14

- 9) Nos vuelve a pedir que operación queremos realizar y elegimos la resta de matrices

```
1. Calcular la suma de matrices.
2. Calcular la resta de matrices
3. Calcular la traspuesta
4. Calcular la multiplicacion de matrices
5. Salir.
```

Introduzca opción deseada 2

La resta de las matrices es la siguiente:

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

10) Volvemos a elegir la opción deseada y nos dará el resultado

```
Traspuesta de la matriz a
Visualizar la matriz normal
  1  2  3  4  5  6  7
  1  2  3  4  5  6  7
  1  2  3  4  5  6  7
  1  2  3  4  5  6  7
  1  2  3  4  5  6  7
  1  2  3  4  5  6  7
  1  2  3  4  5  6  7

Visualizar la traspuesta de la matriz
  1  1  1  1  1  1  1
  2  2  2  2  2  2  2
  3  3  3  3  3  3  3
  4  4  4  4  4  4  4
  5  5  5  5  5  5  5
  6  6  6  6  6  6  6
  7  7  7  7  7  7  7

Traspuesta de la matriz b
Visualizar la matriz normal
  1  2  3  4  5  6  7
  1  2  3  4  5  6  7
  1  2  3  4  5  6  7
  1  2  3  4  5  6  7
  1  2  3  4  5  6  7
  1  2  3  4  5  6  7
  1  2  3  4  5  6  7

Visualizar la traspuesta de la matriz
  1  1  1  1  1  1  1
  2  2  2  2  2  2  2
  3  3  3  3  3  3  3
  4  4  4  4  4  4  4
  5  5  5  5  5  5  5
  6  6  6  6  6  6  6
  7  7  7  7  7  7  7
```

11) Tiempo de ejecución del código

```
Process exited after 521.4 seconds with return value 3221225477
```

Conclusión

Llegando a este punto, hemos aprendido como el sistema operativo se encuentra estructurado en una primera etapa, el desarrollo de procesos y la compresión de hilos de ejecución a través de los cuales, el sistema delega funciones y opera en forma multifuncional.

También estudiamos la forma en la que el sistema gestiona las interrupciones, este se encarga de controlar los accesos al procesador, verificar el estatus de un proceso y determinar su ejecución de acuerdo con el nivel de importancia, cabe destacar que no todas las interrupciones son controladas por el SO, ya que existen interrupciones enmascaradas y que son exclusivas del hardware de nuestro ordenador.

La manera en la que opera nuestra computadora es muy compleja, una vez mas comprender su funcionamiento, nos permite prever las soluciones a ciertos problemas de ejecución, así establecer prioridades al monto de asignar tareas, sobre todo los sistemas multiusuarios.