



Instituto Politécnico Nacional

ESCOM

Practica 3

Administrador de procesos en Linux y Windows

Sistemas Operativos – 2CM17

Integrantes:

Mora Ayala José Antonio

Ramírez Cotonieto Luis Fernando

Torres Carrillo Josehf Miguel Angel

Tovar Jacuinde Rodrigo

Profesor:

Cortes Galicia Jorge

INSTITUTO POLITÉCNICO NACIONAL



INTRODUCCIÓN

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

Como todos sabemos, conforme ha pasado el tiempo las generaciones van cambiando la manera en que experimentan la evolución tecnológica que se ha dado en el mundo desde hace varios años y la cual, hoy continua, pues la tecnología cada día avanza y se van realizando nuevos descubrimientos a diario. Situación que resulta benéfica la mayoría del tiempo, pero también puede llegar a ser contraproducente, pues conforme más acceso y facilidad se tiene a estas tecnologías y lo que conlleva interactuar con las mismas las personas en general han perdido el interés en conocer cómo es que funcionan o que partes son las que componen los dispositivos con los cuales interactúan día a día.

Es por eso que comenzaremos dando un antecedente y conceptos generales y básicos para comprender de una forma más completa esta practica

¿QUÉ ES UN PROCESO?

Cada programa que se ejecuta es un proceso con recursos asignados y gestionado por el kernel. La [™] gestión de procesos comprende la monitorización, detención y cambio de prioridad de los procesos. Generalmente los procesos son gestionados automáticamente por el kernel del S.O. (son creados, ejecutados y detenidos sin la intervención del usuario). Algunas veces los procesos se detendrán por razones desconocidas y será necesario reiniciar el proceso. Otras veces algún proceso se ejecutará descontroladamente malgastando los recursos del sistema, entonces será necesaria una intervención manual del administrador para detener el proceso.

PÁRAMETROS DE UN PROCESO:

PROCESS ID (PID): Cada proceso tiene un número asociado que se le asigna cuando es creado. Los PIDs son números enteros únicos para todos los procesos sistema. **USER ID & GROUP ID:** Cada proceso tiene que tener asociado unos privilegios que limiten el acceso al sistema de ficheros. Estos privilegios quedan determinados por el user ID y group ID del usuario que creo el proceso. **PARENT PROCESS:** Todo proceso es creado por otro proceso, el proceso padre (parent process). El primer proceso iniciado por el kernel cuando el sistema arranca es el programa init. Este proceso tiene el PID 1 y es el padre de todos los procesos del sistema. **PARENT PROCESS ID:** El PID del proceso que inicio el proceso hijo. **ENVIROMENT:** Cada proceso mantiene una lista de variables y sus correspondientes valores. El conjunto de estas variables recibe el nombre de process enviroment. Normalmente el

entorno de un proceso hijo se hereda del proceso padre a menos de que se indique de otra forma. **CURRENT WORKING DIRECTORY:** Cada proceso tiene asociado un directorio por defecto, donde el proceso leerá/escribirá archivos, a menos que se le especifique explícitamente lo contrario. **NICE NUMBER:** Permite al usuario modificar la prioridad de ejecución de un proceso.

COMANDOS FUNDAMENTALES

- **ps:** El comando PS es el que permite informar sobre el estado de los procesos. PS esta basado en el sistema de archivos /proc, es decir, lee directamente la información de los archivos que se encuentran en este directorio. Tiene una gran cantidad de opciones, incluso estas opciones varían dependiendo del estilo en que se use el comando
- **pstree:** El comando pstree muestra los procesos en forma de árbol.
- **kill:** El comando kill, que literalmente quiere decir matar, sirve no solo para matar o terminar procesos sino principalmente para enviar señales (signals) a los procesos. La señal por default (cuando no se indica ninguna es terminar o matar el proceso), y la sintaxis es kill PID, siendo PID el número de ID del proceso. Asi por ejemplo, es posible enviar una señal de STOP al proceso y se detendrá su ejecución, después cuando se quiera mandar una señal de Continuar y el proceso continuara desde donde se quedo.
- **top:** Muestra en tiempo real el estado de los procesos del sistema
- **htop:** Programa que administra todos los procesos activos en tiempo real

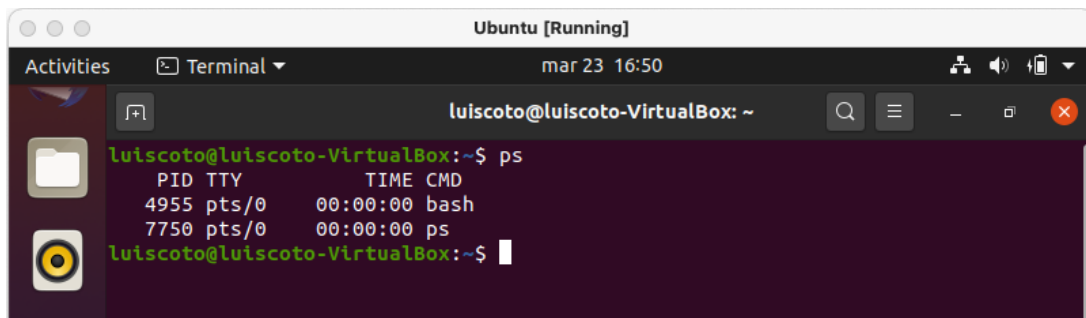
DESARROLLO EXPERIMENTAL

Sección Linux:

1. Introduzca los siguientes comandos a través de la consola del sistema operativo

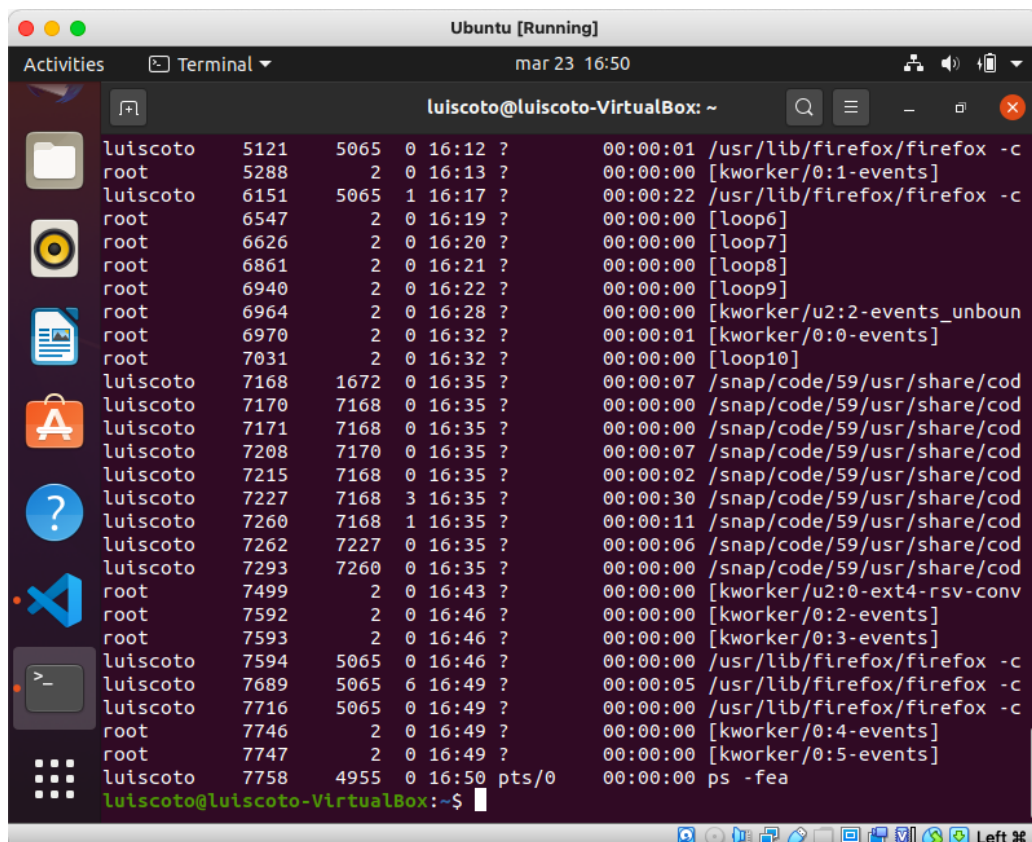
Linux:

Ps



```
Ubuntu [Running]
Activities Terminal mar 23 16:50
luiscoto@luiscoto-VirtualBox: ~
luiscoto@luiscoto-VirtualBox:~$ ps
  PID TTY          TIME CMD
 4955 pts/0    00:00:00 bash
 7750 pts/0    00:00:00 ps
luiscoto@luiscoto-VirtualBox:~$
```

ps -fea



```
Ubuntu [Running]
Activities Terminal mar 23 16:50
luiscoto@luiscoto-VirtualBox: ~
luiscoto 5121 5065 0 16:12 ? 00:00:01 /usr/lib/firefox/firefox -c
root 5288 2 0 16:13 ? 00:00:00 [kworker/0:1-events]
luiscoto 6151 5065 1 16:17 ? 00:00:22 /usr/lib/firefox/firefox -c
root 6547 2 0 16:19 ? 00:00:00 [loop6]
root 6626 2 0 16:20 ? 00:00:00 [loop7]
root 6861 2 0 16:21 ? 00:00:00 [loop8]
root 6940 2 0 16:22 ? 00:00:00 [loop9]
root 6964 2 0 16:28 ? 00:00:00 [kworker/u2:2-events_unboun
root 6970 2 0 16:32 ? 00:00:01 [kworker/0:0-events]
root 7031 2 0 16:32 ? 00:00:00 [loop10]
luiscoto 7168 1672 0 16:35 ? 00:00:07 /snap/code/59/usr/share/cod
luiscoto 7170 7168 0 16:35 ? 00:00:00 /snap/code/59/usr/share/cod
luiscoto 7171 7168 0 16:35 ? 00:00:00 /snap/code/59/usr/share/cod
luiscoto 7208 7170 0 16:35 ? 00:00:07 /snap/code/59/usr/share/cod
luiscoto 7215 7168 0 16:35 ? 00:00:02 /snap/code/59/usr/share/cod
luiscoto 7227 7168 3 16:35 ? 00:00:30 /snap/code/59/usr/share/cod
luiscoto 7260 7168 1 16:35 ? 00:00:11 /snap/code/59/usr/share/cod
luiscoto 7262 7227 0 16:35 ? 00:00:06 /snap/code/59/usr/share/cod
luiscoto 7293 7260 0 16:35 ? 00:00:00 /snap/code/59/usr/share/cod
root 7499 2 0 16:43 ? 00:00:00 [kworker/u2:0-ext4-rsv-conv
root 7592 2 0 16:46 ? 00:00:00 [kworker/0:2-events]
root 7593 2 0 16:46 ? 00:00:00 [kworker/0:3-events]
luiscoto 7594 5065 0 16:46 ? 00:00:00 /usr/lib/firefox/firefox -c
luiscoto 7689 5065 6 16:49 ? 00:00:05 /usr/lib/firefox/firefox -c
luiscoto 7716 5065 0 16:49 ? 00:00:00 /usr/lib/firefox/firefox -c
root 7746 2 0 16:49 ? 00:00:00 [kworker/0:4-events]
root 7747 2 0 16:49 ? 00:00:00 [kworker/0:5-events]
luiscoto 7758 4955 0 16:50 pts/0 00:00:00 ps -fea
luiscoto@luiscoto-VirtualBox:~$
```

¿Qué información le proporcionan los comandos anteriores?

Estos dos comandos nos muestran la información de procesos en ejecución del sistema operativo, se nos muestran datos como el id, el id del proceso padre y cuando colocamos el ps-fea se nos muestra la información de esto de una manera mucho mas detallada

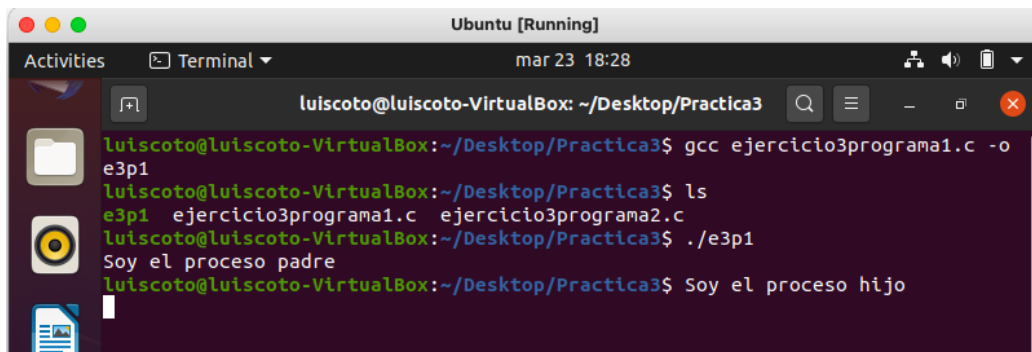
2. A través de la ayuda en línea que proporciona Linux, investigue para que se utiliza el comando ps y mencione las opciones que se pueden utilizar con dicho comando. Además investigue el uso de las llamadas al sistema fork(), execv(), getpid(), getppid() y wait() en la ayuda en línea, mencione que otras funciones similares a execv() existen, reporte sus observaciones.
 - a. **fork():** jd
Fork crea un nuevo proceso duplicando la llamada del proceso, el nuevo proceso es llamado proceso hijo, el proceso que lo llamo es nombrado el proceso padre
 - b. **execv():** jd
es un comando que funciona para ejecutar un programa externo mediante los parámetros de un char denominado argv[] el cual es un puntero
 - c. **getpid():** jd
Nos devuelve el identificador de proceso actual, eso es usado normalmente por rutinas que generan nombres únicos de ficheros temporales
 - d. **getppid():** jd
Nos devuelve el identificador del proceso del padre del proceso actual
 - e. **wait():** jd
la función hace que el proceso espere por la finalización de cualquier hijo y retorna el PID del proceso hijo

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);
    }
}
```

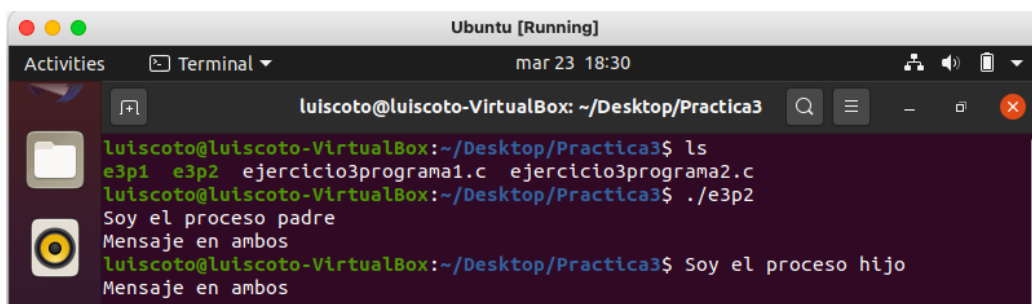
```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(void)
{
    int id_proc;
    id_proc=fork();
    if (id_proc == 0)
    {
        printf("Soy el proceso hijo\n");
    }
    else
    {
        printf("Soy el proceso padre\n");
    }
    printf("Mensaje en ambos\n");
    exit(0);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

Programa 1



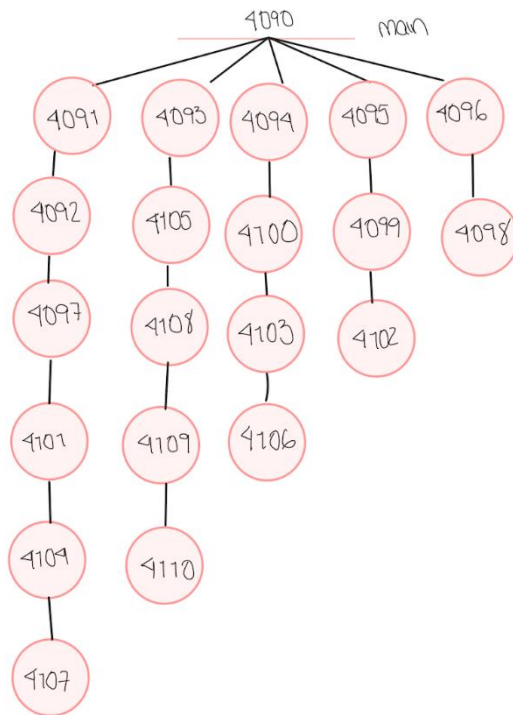
```
Ubuntu [Running]
Activities Terminal mar 23 18:28
luiscoto@luiscoto-VirtualBox: ~/Desktop/Practica3
luiscoto@luiscoto-VirtualBox:~/Desktop/Practica3$ gcc ejercicio3programa1.c -o e3p1
luiscoto@luiscoto-VirtualBox:~/Desktop/Practica3$ ls
e3p1 ejercicio3programa1.c ejercicio3programa2.c
luiscoto@luiscoto-VirtualBox:~/Desktop/Practica3$ ./e3p1
Soy el proceso padre
luiscoto@luiscoto-VirtualBox:~/Desktop/Practica3$ Soy el proceso hijo
```

Programa 2



```
Ubuntu [Running]
Activities Terminal mar 23 18:30
luiscoto@luiscoto-VirtualBox: ~/Desktop/Practica3
luiscoto@luiscoto-VirtualBox:~/Desktop/Practica3$ ls
e3p1 e3p2 ejercicio3programa1.c ejercicio3programa2.c
luiscoto@luiscoto-VirtualBox:~/Desktop/Practica3$ ./e3p2
Soy el proceso padre
Mensaje en ambos
luiscoto@luiscoto-VirtualBox:~/Desktop/Practica3$ Soy el proceso hijo
Mensaje en ambos
```

- Programe una aplicación que cree el árbol de procesos mostrado al final de este documento. Para cada uno de los procesos hijos creados (por copia exacta de código) 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.



```

tony      4348  4064  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4349  4348  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4350  4348  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4351  4349  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4352  4348  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4353  4348  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4354  4348  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4355  4351  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4356  4354  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4357  4353  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4358  4352  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4359  4355  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4360  4357  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4361  4359  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4362  4358  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4363  4350  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4364  4361  7 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4365  4362  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4366  4363  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4367  4366  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4368  4367  6 21:31 pts/0    00:00:00 /home/tony/arbol
tony      4378  4278  0 21:31 pts/2    00:00:00 ps -fea
tony@Tony:~$

```

```

cd "/home/tony/" && gcc prueba2.c -o prueba2 && "/home/tony/"prueba2
tony@Tony:~$ cd "/home/tony/" && gcc prueba2.c -o prueba2 && "/home/tony/"prueba2
4098 Comienzo:
    4098 genero a: 4091
    4098 genero a: 4093
        4091 genero a: 4092
    4098 genero a: 4094
    4098 genero a: 4095
    4098 genero a: 4096
        4092 genero a: 4097
    4096 genero a: 4098
    4095 genero a: 4099
    4094 genero a: 4100
        4097 genero a: 4101
    4098 Fin
4096 Fin
        4099 genero a: 4102
        4100 genero a: 4103
            4101 genero a: 4104
        4102 Fin
    4099 Fin
4095 Fin
    4093 genero a: 4105
        4103 genero a: 4106
            4104 genero a: 4107
        4105 genero a: 4108
            4106 Fin
        4103 Fin
    4100 Fin
4094 Fin
            4107 Fin
            4104 Fin
            4101 Fin
        4097 Fin
    4092 Fin
        4100 genero a: 4109
    4091 Fin
            4109 genero a: 4110
            4110 Fin
        4109 Fin
    4100 Fin
    4105 Fin
    4093 Fin
4098 terminated
tony@Tony:~$

```



```

1 #include<unistd.h>
2 #include<sys/wait.h>
3 #include<stdio.h>
4 int main()
5 {
6     int i,j,k,x,s;
7
8     for ( i = 0; i <5; i++)
9     {
10         if(fork()==0){
11             if(i==0){
12                 for(j=0;j<5;j++){
13                     if(fork()!=0)break;
14                 }
15             }
16             else if(i==1){
17                 for(j=0;j<4;j++){
18                     if(fork()!=0)break;
19                 }
20             }
21             else if(i==2){
22                 for(j=0;j<3;j++){
23                     if(fork()!=0)break;
24                 }
25             }
26             else if(i==3){
27                 for(j=0;j<2;j++){
28                     if(fork()!=0)break;
29                 }
30             }
31             else if (i==4){
32                 for(j=0;j<1;j++){
33                     if(fork()!=0)break;
34                 }
35             }
36             break;
37         }
38     }
39 }
40
41 while (1);
42 return 0;
43

```

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

int level = 1;
char const offsets[] = "t|t|t|t|t|t|t|t";

pid_t Crea_proceso_hijo(int(*child_fn)()) {

    fflush(stdout);
    fflush(stderr);

    pid_t child_pid = fork();
    switch(child_pid) {
    case 0: // Proceso hijo
        ++level;
        exit(child_fn());
    case -1: // En caso de que el fork() falle
        abort();
    default: // Proceso padre
        printf("%s %u genero a: %u\n", level, offsets, (unsigned) getpid(), (unsigned) child_pid);
        return child_pid;
    }
}

void Espera_hijo() { ...

int p21() {return 0;}
int p20() {return 0;}
int p18() {return 0;}
int p15() {return 0;}
int p11() {return 0;}

int p19(){...

int p16(){...
int p12(){...
int p7(){...
int p2(){...

int p17(){...
int p13(){...
int p8(){...
int p3(){...

int p14(){...
int p9(){...
int p4(){...

int p10(){...
int p5(){...
int p6(){...

}

int p1() {
    printf("%u Comienzo:\n", (unsigned) getpid());
    Crea_proceso_hijo(p2);
    Crea_proceso_hijo(p3);
    Crea_proceso_hijo(p4);
    Crea_proceso_hijo(p5);
    Crea_proceso_hijo(p6);
    Espera_hijo();
    Espera_hijo();
}
```

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

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

#define FILAS_MATRIZ_B 7
#define COLUMNAS_MATRIZ_B 7
#define FILAS_MATRIZ_A 7
#define COLUMNAS_MATRIZ_A 7
mode_t mode = S_IRUSR | S_IWUSR;

char cadena4[4];
char cadena3[3];
char cadena2[2];
char coco[7];

int x,i,fichero;
int producto[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];
int suma[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];
int repo[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];

void odio();

int main(int argc, char* argv[]){
    int matrizA[FILAS_MATRIZ_A][COLUMNAS_MATRIZ_A] = {
        {3, 2, 8, 8, 1, 6, 4},
        {20, 8, 4, 5, 20, 5, 9},
        {15, 16, 16, 4, 2, 4, 4},
        {10, 14, 15, 1, 8, 8, 1},
        {9, 1, 5, 5, 9, 5, 6},
        {9, 2, 5, 6, 7, 15, 1},
        {14, 1, 6, 5, 1, 14, 8},
    };
    int matrizB[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B] = {
        {3, 2, 8, 8, 1, 6, 4},
        {1, 8, 4, 5, 7, 5, 9},
        {8, 6, 8, 4, 2, 4, 4},
        {7, 4, 4, 1, 6, 6, 1},
    };
```

```
pid_t pid1, pid2, pid3, pid4, pid5;
int status1, status2, status3, status4, status5;
int j=0;

if((pid1 = fork()) == 0){ //MULTIPLICACION DE MATRICES

    printf("soy el primer hijo (%d, hijo de %d)\n",getpid(), getppid());
    if (COLUMNAS_MATRIZ_A != FILAS_MATRIZ_B) {
        printf("Columnas de matriz A deben ser igual a filas de matriz B");
        return 0;
    }

    for (int a = 0; a < COLUMNAS_MATRIZ_B; a++) {
        for (int i = 0; i < FILAS_MATRIZ_A; i++) {
            int suma = 0;
            for (int j = 0; j < COLUMNAS_MATRIZ_A; j++) {
                suma += matrizA[i][j] * matrizB[j][a];
            }
            producto[i][a] = suma;
        }
    }
    printf("IMPRIENDO EL PRODUCTO\n");
    for (int i = 0; i < FILAS_MATRIZ_B; i++) {
        for (int j = 0; j < COLUMNAS_MATRIZ_B; j++) {
            repo[i][j] = producto[i][j];
        }
        printf("\n");
    }

    fichero = creat("/home/novachrono/practica3/prueba.txt", mode);
    fichero = open ("/home/novachrono/practica3/prueba.txt", O_WRONLY);
    for (x = 0; x < 7; ++x)
    {
        odio();
    }
}
```

```
for(int i=0;i<FILAS_MATRIZ_B;i++){
    printf(" ");
    for(int j=0;j<COLUMNAS_MATRIZ_B;j++){
        repo[i][j] = resta[i][j];
    }
    printf("\n");
}
fichero = creat("/home/novachrono/practica3/resta.txt", mode);
fichero = open ("/home/novachrono/practica3/resta.txt", O_WRONLY);
for( x = 0; x < 7; ++x)
{
    odio();
}
//write(fichero, &coco, sizeof(coco));
close(fichero);
sleep(10);
}
else{
    if(((pid=fork()))==0){ //MATRIZ TRANSPUESTA
        int n;
        printf("soy el cuarto hijo (%d, hijo de %d)\n",getpid(), getppid());

        printf("elijas de que matriz quiere su transpuesta: \n");
        printf("1 para la primera\n");
        printf("2 para la segunda\n");
        scanf("%d", &n);

        if(n==1){
            int matrix[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];
            for (int i = 0; i < COLUMNAS_MATRIZ_A; i++)
            {
                for (int j = 0; j < FILAS_MATRIZ_A; j++)
                {
                    matrix[i][j] = matrizA[j][i];
                }
            }
            //IMPRIMIR PRIMER TRANSPUESTA
            for(int i=0;i<FILAS_MATRIZ_A;i++){
                printf(" ");
                for(int j=0;j<COLUMNAS_MATRIZ_A;j++){
```

```
//IMPRIMIR PRIMER TRANSPUESTA
for(int i=0;i<FILAS_MATRIZ_A;i++){
    printf(" ");
    for(int j=0;j<COLUMNAS_MATRIZ_A;j++){
        printf(" ");
        repo[i][j] = matrix[i][j];
    }
    printf("\n");
}
fichero = creat("/home/novachrono/practica3/trans.txt", mode);
fichero = open ("/home/novachrono/practica3/trans.txt", 0_WRONLY);
for (x = 0; x < 7; ++x)
{
    odio(i);
}
close(fichero);
}

if(n==2){
    int matrix2[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];
    for (int i = 0; i < COLUMNAS_MATRIZ_B; i++)
    {
        for (int j = 0; j < FILAS_MATRIZ_B; j++)
        {
            matrix2[i][j] = matrixB[j][i];
        }
    }
}

//IMPRIMIR SEGUNDA TRANSPUESTA
for(int i=0;i<FILAS_MATRIZ_B;i++){
    printf(" ");
    for(int j=0;j<COLUMNAS_MATRIZ_B;j++){
        printf("%d",matrix2[i][j] );
        repo[i][j] = matrix2[i][j];
    }
    printf("\n");
}
fichero = creat("/home/novachrono/practica3/trans.txt", mode);
fichero = open ("/home/novachrono/practica3/trans.txt", 0_WRONLY);
for (x = 0; x < 7; ++x)
```

```

}
else{
if((pid5=fork())==0){
printf("soy el quinto hijo (%d, hijo de %d)\n",getpid(), getppid());
int fd,n;
char c;
printf("elija que archivo quiere leer: \n");
printf("1 para producto\n");
printf("2 para suma\n");
printf("3 para resta\n");
printf("4 para transpuesta\n");
scanf("%d", &n);
if(n==1){
fd = open("prueba.txt", O_RDONLY);
if(fd!=-1){
while(read(fd,&c,sizeof(c)!=0)){
printf("%c",c);
}
close(fd);
}
}
if(n==2){
fd = open("suma.txt", O_RDONLY);
if(fd!=-1){
while(read(fd,&c,sizeof(c)!=0)){
printf("%c",c);
}
close(fd);
}
}
if(n==3){
fd = open("resta.txt", O_RDONLY);
if(fd!=-1){
while(read(fd,&c,sizeof(c)!=0)){
printf("%c",c);
}
close(fd);
}
}
}
}
}

```

```

void odio(){
for (i = 0; i < 7; ++i){
if (repo[x][i] > 99){
sprintf(cadena4, "%d", repo[x][i]);
write(fichero, &cadena4, sizeof(cadena4));
memset(cadena4, 0, 4);
}
if(repo[x][i] < 100 && repo[x][i] > 9){
sprintf(cadena3, "%d", repo[x][i]);
cadena3[2] = ',';
write(fichero, &cadena3, sizeof(cadena3));
memset(cadena3, 0, 3);
}
if(repo[x][i] < 10){
sprintf(cadena2, "%d, ", repo[x][i]);
write(fichero, &cadena2, sizeof(cadena2));
memset(cadena2, 0, 2);
}
if (i == 6){
write(fichero, " \n", 2);
}
}
}

```



main



main.c



prueba.txt



resta.txt



sopa.c



suma.txt



trans.txt

```

~/practica3/resta.txt - Mousepad
Fichero Editar Búsqueda Ver Documento Ayuda
0,0,0,0,0,0,0,
19,0,0,0,13,0,0,
7,10,8,0,0,0,0,
3,10,11,0,2,2,0,
9,0,2,0,1,0,0,
2,0,0,0,3,11,0,
7,0,0,0,0,8,0,

```

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 a ejecutar. Observe su funcionamiento y experimente con el código.

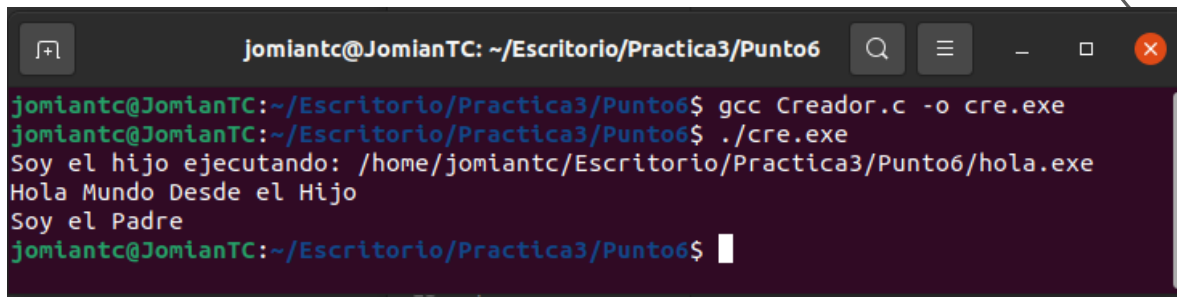
```
/* Programa creador de un nuevo proceso */
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main()
{
    pid_t pid;
    char *argv[3];
    argv[0]="/usuario/hola"; /*cambiar por ruta propia*/
    argv[1]="Desde el Hijo";
    argv[2]=NULL;
    if((pid=fork())==1)
        printf("Error al crear el proceso hijo\n");
    if(pid==0)
    {
        printf("Soy el hijo ejecutando: %s\n", argv[0]);
        execv(argv[0],argv);
    }
    else
    {
        wait(0);
        printf("Soy el Padre\n");
        exit(0);
    }
}
```

```
/* hola.c Programa que será invocado */
#include<stdio.h>
int main(int argc, char *argv[])
{
    char mensaje[100];
    strcpy(mensaje,"Hola Mundo ");
    strcat(mensaje,argv[1]);
    printf("%s\n",mensaje);
    exit(0);
}
```

Obtenga el ejecutable hola del programa hola.c antes de ejecutar el programa creador de un nuevo proceso.

Ejecución de programa

A terminal window titled 'jomiantc@JomianTC: ~/Escritorio/Practica3/Punto6'. The prompt is 'jomiantc@JomianTC:~/Escritorio/Practica3/Punto6\$'. The user enters 'gcc Creador.c -o cre.exe'. The prompt changes to 'jomiantc@JomianTC:~/Escritorio/Practica3/Punto6\$' and the user enters './cre.exe'. The program outputs 'Soy el hijo ejecutando: /home/jomiantc/Escritorio/Practica3/Punto6/hola.exe' and 'Hola Mundo Desde el Hijo'. The prompt returns to 'jomiantc@JomianTC:~/Escritorio/Practica3/Punto6\$' and the user enters 'Soy el Padre'.

```
jomiantc@JomianTC:~/Escritorio/Practica3/Punto6$ gcc Creador.c -o cre.exe
jomiantc@JomianTC:~/Escritorio/Practica3/Punto6$ ./cre.exe
Soy el hijo ejecutando: /home/jomiantc/Escritorio/Practica3/Punto6/hola.exe
Hola Mundo Desde el Hijo
Soy el Padre
jomiantc@JomianTC:~/Escritorio/Practica3/Punto6$
```

7. Programe una aplicación que cree un proceso hijo a partir de un proceso padre, el hijo creado a su vez creará tres procesos hijos más. Cada uno de los tres procesos generados ejecutará tres programas diferentes mediante sustitución de código, el primer programa resolverá una ecuación algebraica de segundo grado mediante la fórmula general, el segundo programa mostrará la serie de Fibonacci para un número N, y el tercer programa obtendrá la multiplicación de dos matrices de 7x7 elementos de tipo entero. Observe el funcionamiento de su programa detalladamente y responda la siguiente pregunta ¿es posible un funcionamiento 100% concurrente de su aplicación? Explique porque sí o no es 100% concurrente su aplicación.

Programa Padre.

```
1  #include<unistd.h>
2  #include<sys/wait.h>
3  #include<stdio.h>
4  #include<stdlib.h>
5
6  int main(){
7
8      pid_t i,j,k,l;
9
10     char *argv[3];
11
12     argv[0]="/home/jomiantc/Escritorio/Practica3/Punto7/h11.exe";
13     argv[1]="/home/jomiantc/Escritorio/Practica3/Punto7/h12.exe";
14     argv[2]="/home/jomiantc/Escritorio/Practica3/Punto7/h13.exe";
15     argv[3]=NULL;
16
17     if((i=fork())== -1)
18         printf("Error al crear el proceso hijo\n");
19
20     if (i==0){
21
22         if((j=fork())== -1)
23             printf("Error al crear el proceso hijo\n");
24
25         if (j==0){
26
27             printf("\n");
28             execv(argv[0],argv);
29         }
30
31         else{
32
33             wait(0);
34
35             if((k=fork())== -1)
36                 printf("Error al crear el proceso hijo\n");
37
38             if (k==0){
39
40                 printf("\n");
41                 execv(argv[1],argv);
42             }
43
44             else{
45
46                 wait(0);
47
48                 if((l=fork())== -1)
49                     printf("Error al crear el proceso hijo\n");
50
51                 if (l==0){
52
53                     printf("\n");
54                     execv(argv[2],argv);
55                 }
56
57                 else{
58
59                     wait(0);
60                     exit(0);
61                 }
62             }
63         }
64     }
65
66     else{
67
68         wait(0);
69         exit(0);
70     }
71 }
```

Programa hijo11.c

```
1  #include <stdio.h> //para compilar usar exit(0); cc hijo11.c -lm -o h11.exe
2  #include <stdlib.h>
3  #include <math.h>
4
5  int a, b, c, resultado;
6
7  int main(int argc, char *argv[]){
8
9      printf("Programa que resuelve la siguiente ecuacion cuadratica: 5x^2-20x=15 = 0 \n");
10
11      a = 5;
12      b = -20;
13      c = 15;
14
15      resultado = (b*b)-(4*a*c);
16      resultado = sqrt(resultado);
17      resultado = ((-1)*b)-resultado;
18      resultado = resultado/(2*a);
19
20      printf("X1 es igual a: %d\n", resultado);
21
22      resultado = (b*b)-(4*a*c);
23      resultado = sqrt(resultado);
24      resultado = ((-1)*b)+resultado;
25      resultado = resultado/(2*a);
26
27      printf("X2 es igual a: %d\n\n", resultado);
28
29      exit(0);
30 }
```


Programa hijo13.c

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  #define FILAS_MATRIZ_B 7
5  #define COLUMNAS_MATRIZ_B 7
6  #define FILAS_MATRIZ_A 7
7  #define COLUMNAS_MATRIZ_A 7
8
9  int main(int argc, char *argv[]){
10
11     printf("programa que imprime un producto de matrices de 7x7\n");
12
13     int matrizA[FILAS_MATRIZ_A][COLUMNAS_MATRIZ_A] = {
14         {3, 2, 1, 3, 2, 1, 3},
15         {1, 1, 3, 1, 1, 3, 1},
16         {0, 2, 1, 0, 2, 1, 0},
17         {3, 2, 1, 3, 2, 1, 3},
18         {1, 1, 3, 1, 1, 3, 1},
19         {0, 2, 1, 0, 2, 1, 0},
20         {3, 2, 1, 3, 2, 1, 3},
21     };
22
23     int matrizB[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B] = {
24         {2, 1, 2, 1, 2, 1, 2},
25         {1, 0, 1, 0, 1, 0, 1},
26         {3, 2, 3, 2, 3, 2, 3},
27         {2, 1, 2, 1, 2, 1, 2},
28         {1, 0, 1, 0, 1, 0, 1},
29         {3, 2, 3, 2, 3, 2, 3},
30         {2, 1, 2, 1, 2, 1, 2},
31     };
32
33     int producto[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];
34
35     for (int a = 0; a < COLUMNAS_MATRIZ_B; a++) {
36
37         for (int i = 0; i < FILAS_MATRIZ_A; i++) {
38
39             int suma = 0;
40
41             for (int j = 0; j < COLUMNAS_MATRIZ_A; j++) {
42
43                 suma += matrizA[i][j] * matrizB[j][a];
44             }
45
46             producto[i][a] = suma;
47         }
48     }
49
50     printf("Imprimiendo producto de matriz\n");
51
52     for (int i = 0; i < FILAS_MATRIZ_B; i++) {
53
54         for (int j = 0; j < COLUMNAS_MATRIZ_B; j++) {
55
56             printf("%d ", producto[i][j]);
57         }
58
59         printf("\n");
60     }
61
62     exit(0);
63 }
```

Programa hijo12.c

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  int n = 10, x = 0, y = 1, z = 1, i;
5
6  int main(int argc, char *argv[]){
7
8      printf("Serie de Fibonacci hasta el decimo numero \n");
9
10     printf("%i, ",z);
11
12     for (i = 0; i < n; i++){
13
14         z = x+y;
15         x = y;
16         y = z;
17         printf(" %d, ", z );
18     }
19
20     printf("\n");
21     exit(0);
22 }
```

Ejecución

```
jomiantc@JomianTC: ~/Escritorio/Practica3/Punto7
jomiantc@JomianTC:~/Escritorio/Practica3/Punto7$ gcc padre.c -o p.exe
jomiantc@JomianTC:~/Escritorio/Practica3/Punto7$ ./p.exe

Programa que resuelve la siguiente ecuacion cuadratica:  $5x^2-20x=15 = 0$ 
X1 es igual a: 1
X2 es igual a: 3

Serie de Fibonacci hasta el decimo numero
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,

programa que imprime un producto de matrices de 7x7
Imprimiendo producto de matriz
28 13 28 13 28 13 28
26 15 26 15 26 15 26
10 4 10 4 10 4 10
28 13 28 13 28 13 28
26 15 26 15 26 15 26
10 4 10 4 10 4 10
28 13 28 13 28 13 28
jomiantc@JomianTC:~/Escritorio/Practica3/Punto7$
```

Es difícil decir si el programa es concurrente o no, de una manera si lo es ya que los procesos de los hijos 11, 12 y 13 se ejecutan 1 a 1 terminando primero uno y luego ejecutando el siguiente hasta terminar pero todo se lleva a cabo mientras que el proceso hijo 1 aún se está ejecutando así que de cierta forma sería concurrente pero si ese proceso no existiera el programa no sería concurrente, aunque algunas opiniones afirman que si lo es ya que aún existe el proceso padre que ejecutaría a los procesos 11, 12 y 13 así que sería concurrente

8. Programe la aplicación desarrollada en el punto 5 de la sección de Linux utilizando esta vez la creación de procesos por sustitución de código.

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

int main()
{
    pid_t pid;
    char *argv[3];
    argv[1]="Desde el Hijo";
    argv[2]=NULL;
    if((pid=fork())==-1)
        printf("Error al crear el proceso hijo\n");
    if(pid==0)
    {
        int n;
        printf("escoja su opcion: \n");
        printf("1 para producto: \n");
        printf("2 para suma: \n");
        printf("3 para resta: \n");
        printf("4 para transpuesta: \n");
        printf("5 para mostrar resultado: \n");
        scanf("%d",&n);
        if (n==1)
        {
            argv[0]="/home/novachrono/practica3/prog2";
            execv(argv[0],argv);
        }
        if (n==2)
        {
            argv[0]="/home/novachrono/practica3/prog3";
            execv(argv[0],argv);
        }
        if (n==3)
        {
            argv[0]="/home/novachrono/practica3/prog4";
            execv(argv[0],argv);
        }
        if (n==4)
        {
            argv[0]="/home/novachrono/practica3/prog5";
            execv(argv[0],argv);
        }
        if (n==5)
        {
            argv[0]="/home/novachrono/practica3/prog6";
            execv(argv[0],argv);
        }
    }
    else
    {
        wait(0);
        printf("Soy el Padre\n");
        exit(0);
    }
}
```

Primer proceso hijo

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

#define FILAS_MATRIZ_B 7
#define COLUMNAS_MATRIZ_B 7
#define FILAS_MATRIZ_A 7
#define COLUMNAS_MATRIZ_A 7
mode_t mode = S_IRUSR | S_IWUSR;

char cadena4[4];
char cadena3[3];
char cadena2[2];
char coco[7];

int x,i,fichero;
int producto[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];
int suma[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];
int repo[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];

void odio();

int main(void) {

    int matrizA[FILAS_MATRIZ_A][COLUMNAS_MATRIZ_A] = {
        {3, 2, 8, 8, 1, 6, 4},
        {20, 8, 4, 5, 20, 5, 9},
        {15, 16, 16, 4, 2, 4, 4},
        {10, 14, 15, 1, 8, 8, 1},
        {9, 1, 5, 5, 9, 5, 6},
        {9, 2, 5, 6, 7, 15, 1},
        {14, 1, 6, 5, 1, 14, 8},
    };
};
```

```
int matrizB[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B] = {
    {3, 2, 8, 8, 1, 6, 4},
    {1, 8, 4, 5, 7, 5, 9},
    {8, 6, 8, 4, 2, 4, 4},
    {7, 4, 4, 1, 6, 6, 1},
    {0, 1, 3, 5, 8, 5, 6},
    {7, 2, 5, 6, 4, 4, 1},
    {7, 1, 6, 5, 1, 6, 8},
};

for (int a = 0; a < COLUMNAS_MATRIZ_B; a++) {
    for (int i = 0; i < FILAS_MATRIZ_A; i++) {
        int suma = 0;
        for (int j = 0; j < COLUMNAS_MATRIZ_A; j++) {
            suma += matrizA[i][j] * matrizB[j][a];
        }
        producto[i][a] = suma;
    }
}

printf("IMPRIENDO EL PRODUCTO\n");
for (int i = 0; i < FILAS_MATRIZ_B; i++) {
    for (int j = 0; j < COLUMNAS_MATRIZ_B; j++) {
        repo[i][j] = producto[i][j];
    }
    printf("\n");
}

fichero = creat("/home/novachrono/practica3/prueba.txt", mode);
fichero = open ("/home/novachrono/practica3/prueba.txt", O_WRONLY);
for (x = 0; x < 7; ++x)
{
    odio();
}
//write(fichero, &coco, sizeof(coco));
close(fichero);
}
```

Segundo Proceso hijo

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

#define FILAS_MATRIZ_B 7
#define COLUMNAS_MATRIZ_B 7
#define FILAS_MATRIZ_A 7
#define COLUMNAS_MATRIZ_A 7
mode_t mode = S_IRUSR | S_IWUSR;

char cadena4[4];
char cadena3[3];
char cadena2[2];
char coco[7];

int x,i,fichero;
int producto[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];
int suma[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];
int repo[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];

void odio();

int main(void) {

    int matrizA[FILAS_MATRIZ_A][COLUMNAS_MATRIZ_A] = {
        {3, 2, 8, 8, 1, 6, 4},
        {20, 8, 4, 5, 20, 5, 9},
        {15, 16, 16, 4, 2, 4, 4},
        {10, 14, 15, 1, 8, 8, 1},
        {9, 1, 5, 5, 9, 5, 6},
        {9, 2, 5, 6, 7, 15, 1},
        {14, 1, 6, 5, 1, 14, 8},
    };
};
```

```
for(int i=0;i<FILAS_MATRIZ_B; i++){
    for(int j=0;j<COLUMNAS_MATRIZ_B;j++){
        suma[i][j] = matrizA[i][j] + matrizB[i][j];
    }
}
//IMPRIME RESULTADO DE LA SUMA
for(int i=0;i<FILAS_MATRIZ_B;i++){
    printf(" ");
    for(int j=0;j<COLUMNAS_MATRIZ_B;j++){
        repo[i][j] = suma[i][j];
    }
    printf("\n");
}

fichero = creat("/home/novachrono/practica3/suma.txt", mode);
fichero = open ("/home/novachrono/practica3/suma.txt", O_WRONLY);
for (x = 0; x < 7; ++x)
{
    odio();
}
//write(fichero, &coco, sizeof(coco));
close(fichero);
}

void odio(){
    for (i = 0; i < 7; ++i){
        if (repo[x][i] > 99){
            sprintf(cadena4, "%d,", repo[x][i]);
            write(fichero, &cadena4, sizeof(cadena4));
            memset(cadena4, 0, 4);
        }
    }

    if(repo[x][i] < 100 && repo[x][i] > 9){
```

Tercer proceso hijo

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

#define FILAS_MATRIZ_B 7
#define COLUMNAS_MATRIZ_B 7
#define FILAS_MATRIZ_A 7
#define COLUMNAS_MATRIZ_A 7
mode_t mode = S_IRUSR | S_IWUSR;

char cadena4[4];
char cadena3[3];
char cadena2[2];
char coco[7];

int x,i,fichero;
int producto[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];
int suma[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];
int repo[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];

void odio();

int main(void) {

    int matrizA[FILAS_MATRIZ_A][COLUMNAS_MATRIZ_A] = {
        {3, 2, 8, 8, 1, 6, 4},
        {20, 8, 4, 5, 20, 5, 9},
        {15, 16, 16, 4, 2, 4, 4},
        {10, 14, 15, 1, 8, 8, 1},
        {9, 1, 5, 5, 9, 5, 6},
        {9, 2, 5, 6, 7, 15, 1},
        {14, 1, 6, 5, 1, 14, 8},
    };
};
```

```
int resta[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];

for(int i=0;i<FILAS_MATRIZ_B;i++){
    for(int j=0;j<COLUMNAS_MATRIZ_B;j++){
        resta[i][j] = matrizA[i][j] - matrizB[i][j];
    }
}
//IMPRIME RESULTADO DE LA RESTA
for(int i=0;i<FILAS_MATRIZ_B;i++){
    printf(" ");
    for(int j=0;j<COLUMNAS_MATRIZ_B;j++){
        repo[i][j] = resta[i][j];
    }
    printf("\n");
}
fichero = creat("/home/novachrono/practica3/resta.txt", mode);
fichero = open ("/home/novachrono/practica3/resta.txt", O_WRONLY);
for (x = 0; x < 7; ++x)
{
    odio();
}
//write(fichero, &coco, sizeof(coco));
close(fichero);
}

void odio(){
    for (i = 0; i < 7; ++i){
        if (repo[x][i] > 99){
            sprintf(cadena4, "%d,", repo[x][i]);
            write(fichero, &cadena4, sizeof(cadena4));
            memset(cadena4, 0, 4);
        }

        if(repo[x][i] < 100 && repo[x][i] > 9){

```


Cuarto proceso hijo

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

#define FILAS_MATRIZ_B 7
#define COLUMNAS_MATRIZ_B 7
#define FILAS_MATRIZ_A 7
#define COLUMNAS_MATRIZ_A 7
mode_t mode = S_IRUSR | S_IWUSR;

char cadena4[4];
char cadena3[3];
char cadena2[2];
char coco[7];

int x,i,fichero;
int producto[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];
int suma[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];
int repo[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];

void odio();

int main(void) {

    int matrizA[FILAS_MATRIZ_A][COLUMNAS_MATRIZ_A] = {
        {3, 2, 8, 8, 1, 6, 4},
        {20, 8, 4, 5, 20, 5, 9},
        {15, 16, 16, 4, 2, 4, 4},
        {10, 14, 15, 1, 8, 8, 1},
        {9, 1, 5, 5, 9, 5, 6},
        {9, 2, 5, 6, 7, 15, 1},
        {14, 1, 6, 5, 1, 14, 8},
    };
};
```

```
int n;
printf("soy el cuarto hijo (%d, hijo de %d)\n",getpid(), getppid());

printf("elija de que matriz quiere su transpuesta: \n");
printf("1 para la primera\n");
printf("2 para la segunda\n");
scanf("%d", &n);

if(n==1){
    int matrix[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];
    for (int i = 0; i < COLUMNAS_MATRIZ_A; i++)
    {
        for (int j = 0; j < FILAS_MATRIZ_A; j++)
        {
            matrix[i][j] = matrizA[j][i];
        }
    }
    //IMPRIMIR PRIMER TRANSPUESTA
    for(int i=0;i<FILAS_MATRIZ_A;i++){
        printf(" ");
        for(int j=0;j<COLUMNAS_MATRIZ_A;j++){
            printf(" ");
            repo[i][j] = matrix[i][j];
        }
        printf("\n");
    }
    fichero = creat("/home/novachrono/practica3/trans.txt", mode);
    fichero = open ("/home/novachrono/practica3/trans.txt", O_WRONLY);
    for (x = 0; x < 7; ++x)
    {
        odio();
    }
    close(fichero);
}
```

Quinto proceso hijo

```
int fd,n;
char c;
printf("elija que archivo quiere leer: \n");
printf("1 para producto\n");
printf("2 para suma\n");
printf("3 para resta\n");
printf("4 para transpuesta\n");
scanf("%d", &n);
if(n==1){
    fd = open("prueba.txt", O_RDONLY);
    if(fd!=-1){
        while(read(fd,&c,sizeof(c))!=0){
            printf("%c",c);
        }
        close(fd);
    }
}
if(n==2){
    fd = open("suma.txt", O_RDONLY);
    if(fd!=-1){
        while(read(fd,&c,sizeof(c))!=0){
            printf("%c",c);
        }
        close(fd);
    }
}
if(n==3){
    fd = open("resta.txt", O_RDONLY);
    if(fd!=-1){
        while(read(fd,&c,sizeof(c))!=0){
            printf("%c",c);
        }
        close(fd);
    }
}
```


Sección Windows:

1. Inicie sesión en Windows.
2. Para esta práctica se utilizará el ambiente de programación Dev C/C++.
3. Capture y compile el programa de creación de un nuevo proceso que a continuación se muestra.

```
#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);
}
```

4. Capture y compile el programa que contendrá al proceso hijo que a continuación se muestra.
5. Ejecute el primer código pasando como argumento el nombre del archivo

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

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

ejecutable del segundo código capturado. Observe el funcionamiento del programa, reporte sus observaciones y experimente con el código.

Realizando la Ejecución de Código tal como se indica se puede observar que da diversos warnings, pues en el proceso padre cada que se hace una evaluación o una verificación se da un return; el cual no devuelve nada, de igual forma se observa que al momento de realizar la sustitución del nombre del programa hijo en donde se indica este no es aceptado y no se crea dicho proceso, pues argv[0]; nos proporciona la siguiente línea de comando:

Usar: C:\Users\TONY AYALA\Google

Drive\Escolar\College\CuartoSemestre\Sistemas_Operativos\Tarea3\Programas\Puntos3_4_5
\padre.exe

La cual llega hasta la línea ejecutable del programa padre es por eso que al código padre se le ha realizado esta modificación:

```
#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\n", argv[0]);
    }
}
```

```

strcat(argv[0], "\\Ejecutable.exe");
printf("Usar: %s\n", argv[0]);
}
// Creación proceso hijo
if(!CreateProcess("C:\\Users\\TONY AYALA\\Google Drive\\Escolar\\College\\CuartoSemestre\\Sistemas_Operativos\\Tarea3\\Programas\\Puntos3_4_5\\Ejecutable.exe", NULL, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
{
printf( "Fallo al invocar CreateProcess (%d)\n", GetLastError() );
}
// Proceso padre
printf("Soy el padre\n");
WaitForSingleObject(pi.hProcess, INFINITE);
// Terminación controlada del proceso e hilo asociado de ejecución

if(CloseHandle(pi.hProcess) !=0){
printf("El manejador del proceso fue cerrado de forma exitosa\n");}
else{
printf("Algo no sucedio como debia");
}

if(CloseHandle(pi.hThread) != 0)

printf("El manejador de amenazas fue cerrado de forma exitosa\n");

return 0;
}

```

Dando como parámetro a la función creadora del proceso hijo la ruta donde se encuentra dicho proceso y agregando condicionales a aquellos manejadores que cierran los procesos para verificar si fueron cerrados de forma exitosa.

```

Usar: C:\Users\TONY AYALA\Google Drive\Escolar\College\CuartoSemestre\Sistemas_Operativos\Tarea3\Programas\Puntos3_4_5\padre.exe
Usar: C:\Users\TONY AYALA\Google Drive\Escolar\College\CuartoSemestre\Sistemas_Operativos\Tarea3\Programas\Puntos3_4_5\padre.exe\Ejecutable.exe
Soy el padre
Soy el hijo
Que onda
Soy el padre otra vez
El manejador del proceso fue cerrado de forma exitosa
El manejador de amenazas fue cerrado de forma exitosa

```

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

Las diferencias entre estos sistemas son sin lugar a dudas que las funciones y la forma de crear los procesos son tan diferentes entre sí, aun que cumplen la función, por ejemplo en las funciones de Windows tenemos más parámetros en la función `CreateProcess()` mientras que con la función de `Fork()` tiene muchos menos procesos, también es mucho más fácil crear procesos, cerrarlos e identificarlos en Linux que en Windows, con Windows se necesitan más funciones además de que puede tener problemas de compatibilidad con los compiladores o las versiones de Windows

7. Programe 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. CONSEJO: INVESTIGUE LA FUNCIÓN `GetCurrentProcessId()` DEL API WIN32.

Proceso padre

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

int main(int argc, char *argv[])
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    int i;
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

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

    printf("\n\t\t **ARBOL DE PROCESOS**\n");
    printf("\n\n -> Soy el Padre, PID: %d\n", GetCurrentProcessId());
    WaitForSingleObject(pi.hProcess, INFINITE);
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
    return 0;
}
```

Proceso hijo (Nivel 2)

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

int main(void)
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    int i;
    int j;
    printf("\n\t Nivel 2");
    printf("\n\t ---> Soy el hijo, PID: %d\n", GetCurrentProcessId());
    for ( j = 0 ; j < 5 ; j++ )
    {

        ZeroMemory(&si, sizeof(si));
        si.cb = sizeof(si);
        ZeroMemory(&pi, sizeof(pi));

        if ( !CreateProcess(NULL, "hijo3", NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi) )
        {
            printf("Fallo al invocar CreateProcess(%d)\n", GetLastError() );
            return 1;
        }

        WaitForSingleObject(pi.hProcess, INFINITE);
        CloseHandle(pi.hProcess);
        CloseHandle(pi.hThread);
    }

    return 0;
}
```

Proceso hijo (nivel 3)

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

int main()
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    int i;
    int j;
    printf("\n\t\t Nivel 3");
    printf("\n\t\t ---> Soy el hijo, PID: %d\n", GetCurrentProcessId());
    for ( j = 0 ; j < 3 ; j++ )
    {
        ZeroMemory(&si, sizeof(si));
        si.cb = sizeof(si);
        ZeroMemory(&pi, sizeof(pi));

        if( !CreateProcess(NULL, "hijofinal", NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
        {
            printf("Fallo al invocar CreateProcess(%d)\n", GetLastError() );
            return 1;
        }

        WaitForSingleObject(pi.hProcess, INFINITE);
        CloseHandle(pi.hProcess);
        CloseHandle(pi.hThread);
    }
    return 0;
}
```

Proceso hijo final

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

int main()
{
    printf("\n\t\t\t Nivel 4");
    printf("\n\t\t\t ---> Soy el hijo, PID: %d\n", GetCurrentProcessId());
    return 0;
}
```

Ejecución

```
Símbolo del sistema
C:\Users\Luis Coto\Desktop\SO>padre hijo hijo3 hijo3 hijo3
**ARBOL DE PROCESOS**

->Soy el Padre, PID: 4828
  Nivel 2
  --->Soy el hijo, PID: 184
    Nivel 3
    --->Soy el hijo, PID: 2812
      Nivel 4
      --->Soy el hijo, PID: 4256
      Nivel 4
      --->Soy el hijo, PID: 6024
      Nivel 4
      --->Soy el hijo, PID: 2816
    Nivel 3
    --->Soy el hijo, PID: 6044
      Nivel 4
      --->Soy el hijo, PID: 2008
      Nivel 4
      --->Soy el hijo, PID: 4784
      Nivel 4
      --->Soy el hijo, PID: 5896
    Nivel 3
    --->Soy el hijo, PID: 3400
      Nivel 4
      --->Soy el hijo, PID: 1420
      Nivel 4
      --->Soy el hijo, PID: 2568
      Nivel 4
      --->Soy el hijo, PID: 5324
    Nivel 3
    --->Soy el hijo, PID: 1512
      Nivel 4
      --->Soy el hijo, PID: 5944
      Nivel 4
      --->Soy el hijo, PID: 3816
      Nivel 4
      --->Soy el hijo, PID: 656
    Nivel 3
    --->Soy el hijo, PID: 4536
      Nivel 4
      --->Soy el hijo, PID: 1872
      Nivel 4
      --->Soy el hijo, PID: 6032
      Nivel 4
      --->Soy el hijo, PID: 3672
C:\Users\Luis Coto\Desktop\SO>
```

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.

Proceso principal

```
int main(){
    HANDLE hProcess;
    HANDLE hThread;
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    DWORD dwProcessId = 0;
    DWORD dwThreadId = 0;
    ZeroMemory( &si, sizeof(si));
    ZeroMemory( &pi, sizeof(pi));
    BOOL bCreateProcess = FALSE;

    bCreateProcess = CreateProcess(
        "C:\\Users\\Rodrigo\\Desktop\\pruebas\\mult.exe",
        NULL,
        NULL,
        NULL,
        FALSE,
        0,
        NULL,
        NULL,
        &si,
        &pi
    );
```

```
printf("creacion correcta del proceso\n");
printf("process id: %d\n", pi.dwProcessId);
printf("thread id: %d\n", pi.dwThreadId);

WaitForSingleObject(pi.hProcess, INFINITE);
CloseHandle(pi.hThread);
CloseHandle(pi.hProcess);
```



```

hFile = CreateFile(
    "C:\\Users\\Rodrigo\\Desktop\\pruebas\\producto.txt",
    GENERIC_READ|GENERIC_WRITE,
    FILE_SHARE_READ,
    NULL,
    CREATE_NEW,
    FILE_ATTRIBUTE_NORMAL,
    NULL
);
//read file
bFile = ReadFile(
    hFile,
    chBuffer,
    dwNoByteToRead,
    &dwNoByteRead,
    NULL
);
printf("datos del archivo: %s", chBuffer);
CloseHandle(hFile);

```

Proceso de Matriz

```

#define FILAS_MATRIZ_B 7
#define COLUMNAS_MATRIZ_B 7
#define FILAS_MATRIZ_A 7
#define COLUMNAS_MATRIZ_A 7

char cadena4[4];
char cadena3[3];
char cadena2[2];
char coco[7];

int x,i,fichero;
int producto[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];
int suma[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];
int repo[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];

```

```

int matrizA[FILAS_MATRIZ_A][COLUMNAS_MATRIZ_A] = {
    {3, 2, 8, 8, 1, 6, 4},
    {20, 8, 4, 5, 20, 5, 9},
    {15, 16, 16, 4, 2, 4, 4},
    {10, 14, 15, 1, 8, 8, 1},
    {9, 1, 5, 5, 9, 5, 6},
    {9, 2, 5, 6, 7, 15, 1},
    {14, 1, 6, 5, 1, 14, 8},
};

int matrizB[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B] = {
    {3, 2, 8, 8, 1, 6, 4},
    {1, 8, 4, 5, 7, 5, 9},
    {8, 6, 8, 4, 2, 4, 4},
    {7, 4, 4, 1, 6, 6, 1},
    {0, 1, 3, 5, 8, 5, 6},
    {7, 2, 5, 6, 4, 4, 1},
    {7, 1, 6, 5, 1, 6, 8},
};

```

```

for (int a = 0; a < COLUMNAS_MATRIZ_B; a++) {

    for (int i = 0; i < FILAS_MATRIZ_A; i++) {
        int suma = 0;
        for (int j = 0; j < COLUMNAS_MATRIZ_A; j++) {
            suma += matrizA[i][j] * matrizB[j][a];
        }
        producto[i][a] = suma;
    }
}

printf("IMPRIMIENDO EL PRODUCTO\n");
for (int i = 0; i < FILAS_MATRIZ_B; i++) {
    for (int j = 0; j < COLUMNAS_MATRIZ_B; j++) {
        repo[i][j] = producto[i][j];
    }
    printf("\n");
}

for (x = 0; x < 7; ++x)
{
    odio();
}

```

```

void odio(){
    for (i = 0; i < 7; ++i){

        if (repo[x][i] > 99){

            sprintf(cadena4, "%d,", repo[x][i]);
            HANDLE hFile = CreateFile(
                "C:\\Users\\Rodrigo\\Desktop\\pruebas\\producto.txt",
                GENERIC_WRITE,
                FILE_SHARE_READ,
                NULL,
                CREATE_NEW,
                FILE_ATTRIBUTE_NORMAL,
                NULL);
            LPSTR(strText);
            DWORD bytesWritten;
            WriteFile(
                hFile,
                cadena4,
                strlen(cadena4),
                &bytesWritten,
                NULL);
            CloseHandle(hFile);
            memset(cadena4, 0, 4);
        }
    }
}

```

```

if(repo[x][i] < 100 && repo[x][i] > 9){

    sprintf(cadena3, "%d", repo[x][i]);
    cadena3[2] = ',';
    HANDLE hFile = CreateFile(
        "C:\\Users\\Rodrigo\\Desktop\\pruebas\\producto.txt",
        GENERIC_WRITE,
        FILE_SHARE_READ,
        NULL,
        CREATE_NEW,
        FILE_ATTRIBUTE_NORMAL,
        NULL);
    LPSTR(strText);
    DWORD bytesWritten;
    WriteFile(
        hFile,
        cadena3,
        strlen(cadena3),
        &bytesWritten,
        NULL);
    CloseHandle(hFile);
    memset(cadena3, 0, 3);
}

```

```

if(repo[x][i] < 10){
    sprintf(cadena2, "%d, ", repo[x][i]);
    HANDLE hFile = CreateFile(
        "C:\\Users\\Rodrigo\\Desktop\\pruebas\\producto.txt",
        GENERIC_WRITE,
        FILE_SHARE_READ,
        NULL,
        CREATE_NEW,
        FILE_ATTRIBUTE_NORMAL,
        NULL);
    LPSTR(strText);
    DWORD bytesWritten;
    WriteFile(
        hFile,
        cadena2,
        strlen(cadena2),
        &bytesWritten,
        NULL);
    CloseHandle(hFile);
    memset(cadena2, 0, 2);
}

```

```

for(int i=0;i<FILAS_MATRIZ_B; i++){
    for(int j=0;j<COLUMNAS_MATRIZ_B;j++){
        suma[i][j] = matrizA[i][j] + matrizB[i][j];
    }
}
//IMPRIME RESULTADO DE LA SUMA
for(int i=0;i<FILAS_MATRIZ_B;i++){
    printf(" ");
    for(int j=0;j<COLUMNAS_MATRIZ_B;j++){
        repo[i][j] = suma[i][j];
    }
    printf("\n");
}

for (x = 0; x < 7; ++x)
{
    odio();
}

```

```

int resta[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];

    for(int i=0;i<FILAS_MATRIZ_B; i++){
        for(int j=0;j<COLUMNAS_MATRIZ_B;j++){
            resta[i][j] = matrizA[i][j] - matrizB[i][j];
        }
    }
    //IMPRIME RESULTADO DE LA RESTA
    for(int i=0;i<FILAS_MATRIZ_B;i++){
        printf(" ");
        for(int j=0;j<COLUMNAS_MATRIZ_B;j++){
            repo[i][j] = resta[i][j];
        }
        printf("\n");
    }

    for (x = 0; x < 7; ++x)
    {
        odio();
    }

















```

```


int matrix[FILAS_MATRIZ_B][COLUMNAS_MATRIZ_B];
for (int i = 0; i < COLUMNAS_MATRIZ_A; i++)
{
    for (int j = 0; j < FILAS_MATRIZ_A; j++)
    {
        matrix[i][j] = matrizA[j][i];
    }
}
//IMPRIMIR PRIMER TRANSPUESTA
for(int i=0;i<FILAS_MATRIZ_A;i++){
    printf(" ");
    for(int j=0;j<COLUMNAS_MATRIZ_A;j++){
        printf(" ");
        repo[i][j] = matrix[i][j];
    }
    printf("\n");
}
for (x = 0; x < 7; ++x)
{
    odio();
}

```


Prueba de creación de archivos de matrices

	father	28/03/2021 10:33 a. m.	C Source File	4 KB
	father	28/03/2021 10:24 a. m.	Aplicación	43 KB
	leer	28/03/2021 10:51 a. m.	C Source File	4 KB
	leer	28/03/2021 10:31 a. m.	Aplicación	42 KB
	mult	28/03/2021 10:17 a. m.	C Source File	4 KB
	mult	28/03/2021 10:20 a. m.	Aplicación	43 KB
	producto	28/03/2021 10:31 a. m.	Documento de te...	1 KB
	resta	28/03/2021 10:21 a. m.	C Source File	4 KB
	resta	28/03/2021 10:21 a. m.	Aplicación	43 KB
	resta	28/03/2021 10:31 a. m.	Documento de te...	1 KB
	suma	28/03/2021 10:21 a. m.	C Source File	4 KB
	suma	28/03/2021 10:21 a. m.	Aplicación	43 KB
	suma	28/03/2021 10:31 a. m.	Documento de te...	1 KB
	trans	28/03/2021 10:24 a. m.	C Source File	6 KB
	trans	28/03/2021 10:24 a. m.	Aplicación	44 KB
	trans	28/03/2021 10:31 a. m.	Documento de te...	1 KB

Contenido de los archivos creados

 producto: Bloc de notas

Archivo Edición Formato Ver Ayuda

201,119,185,135,117,161,114,
233,187,383,396,303,380,370,
273,284,378,322,219,308,320,
234,251,330,304,241,274,291,
180,101,224,207,154,210,177,
223,126,246,238,186,221,145,
280,129,305,275,135,252,178,

 resta: Bloc de notas

Archivo Edición Formato Ver Ayuda

0,0,0,0,0,0,0,
19,0,0,0,13,0,0,
7,10,8,0,0,0,0,
3,10,11,0,2,2,0,
9,0,2,0,1,0,0,
2,0,0,0,3,11,0,
7,0,0,0,0,8,0,

 suma: Bloc de notas

Archivo Edición Formato Ver Ayuda

6,4,16,16,2,12,8,
21,16,8,10,27,10,18,
23,22,24,8,4,8,8,
17,18,19,2,14,14,2,
9,2,8,10,17,10,12,
16,4,10,12,11,19,2,
21,2,12,10,2,20,16,

 trans: Bloc de notas

Archivo Edición Formato Ver Ayuda

3,20,15,10,9,9,14,
2,8,16,14,1,2,1,
8,4,16,15,5,5,6,
8,5,4,1,5,6,5,
1,20,2,8,9,7,1,
6,5,4,8,5,15,14,
4,9,4,1,6,1,8,

CONCLUSIONES

La realización de esta practica me gusto mucho, debido a que pude tener una interacción mas profunda con los sistemas operativos que día a día manejo, lo cual me permitió comprender mucho mas la forma en que estos funcionan, me pareció muy interesante todo el concepto de los procesos y la forma en que nosotros podemos manipularlos a nuestra conveniencia para cualquier situación que se pueda presentar, como todo una cosa lleva a otra y realmente es lo que sucede con esta práctica, está llena de tantos conceptos tan interesantes que realmente no es difícil o pesado estar leyendo y practicando con los mismos.

-Mora Ayala José Antonio

Esta práctica me resultó realmente interesante pues la mayoría de las funciones que se presentaban en la práctica, no las conocía personalmente yo, incluso no me imaginaba funciones como ejecutar simultaneamente varios procesos, o crear este tipo de arboles de la práctica. Encontre un verdadero reto al momento de utilizar el SO Windows, descubrí que llevo demasiado tiempo sin utilizarlo, por lo que tuve que realizar investigaciones más profundas para este entorno ya que de igual manera, es un poco más complicado debido a que existen funciones que directamente no son tan compatibles, por lo que llego a la conclusión nuevamente que el sistema operativo adecuado para la administración y creación de procesos a nivel consola es el sistema linux.

-Ramírez Cotonieto Luis Fernando

Esta práctica me gusto bastante ya que no sabía que podíamos ejecutar de manera simultánea o casi simultánea 2 programas o procesos a la vez, esto es una manera muy cercana y parecida de la multitarea aunque algo limitada donde podemos ejecutar más de 2 programas a la vez y podemos ejecutar nuevos procesos o cerrar los ya creados únicamente usando la función fork() y execv(), al menos en el sistema operativo de Linux es muy sencillo estas funciones, aunque en Windows es más complejo debido a las incompatibilidades o complejidad de los comandos en los programas, si se quiere hacer uso de estos comandos de manera más sencilla es recomendable que se haga en el entorno de Linux

-Torres Carrillo Josehf Miguel Angel

Después de realizar esta práctica me di cuenta como con alguna función se puede optimizar la velocidad de ejecución de los problemas que implementemos, y que mejor que ocupar la creación de procesos para que estos problemas se ejecuten casi a la vez y obtener un tiempo menor, por otro lado, me resulto un poco complicado de inicio porque tuve que ocupar funciones que desconocía, pero nada que no se solucionara investigando sus manuales y su modo de implementación.

-Rodrigo Tovar Jacuinde