



Instituto Politecnico Nacional

Escuela Superior de Cómputo



Práctica 3

Práctica 3. Administrador de procesos en Linux y Windows (1)

Sistemas Operativos

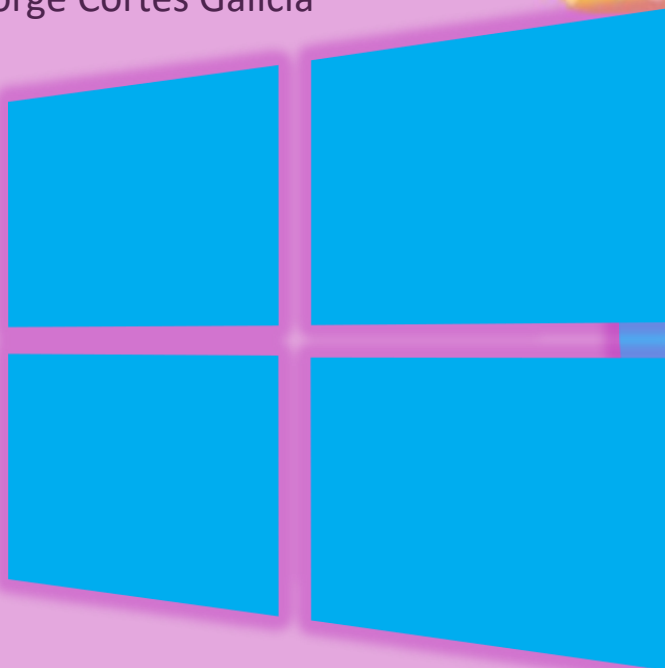
Grupo: 2CM12

Integrantes:

- ⇒ Baldovinos Gutiérrez Kevin
- ⇒ Bocanegra Heziquio Yestlanezi
- ⇒ Castañares Torres Jorge David
- ⇒ Hernández Hernández Rut Esther

Profesor

Jorge Cortes Galicia



Contenido

COPETENCIA.....	3
INTRODUCCIÓN.....	3
DESARROLLO.....	5
Sección Linux.....	5
Procesos impresos	10
IMPRESIÓN.....	11
CODIGO.....	13
CÓDIGO.....	15
Sección Windows	19
Conclusiones	28
Bibliografía	29



COPETENCIA

El alumno aprende a familiarizarse con el administrador de procesos del sistema operativo Linux y Windows a través de la creación de nuevos procesos por copia exacta de código y/o por sustitución de código para el desarrollo de aplicaciones concurrentes sencillas

INTRODUCCIÓN

Un proceso no es más que un programa en ejecución, e incluye los valores actuales del contador de programa, los registros y las variables. Conceptualmente cada uno de estos procesos tiene su propia CPU virtual. Desde luego, en la realidad la verdadera CPU conmuta de un proceso a otro. Un proceso es un concepto manejado por el sistema operativo que consiste en el conjunto formado por:

Las instrucciones de un programa destinadas a ser ejecutadas por el microprocesador.

Su estado de ejecución en un momento dado, esto es, los valores de los registros de la CPU para dicho programa.

Su memoria de trabajo, es decir, la memoria que ha reservado y sus contenidos.

Otra información que permite al sistema operativo su planificación.

Esta definición varía ligeramente en el caso de sistemas operativos multitarea, donde un proceso consta de uno o más hilos, la memoria de trabajo (compartida por todos los hilos) y la información de planificación. Cada hilo consta de instrucciones y estado de ejecución. Los procesos son creados y destruidos por el sistema operativo, así como también este se debe hacer cargo de la comunicación entre procesos, pero lo hace a petición de otros procesos.

El mecanismo por el cual un proceso crea otro proceso se denomina bifurcación (fork). Los nuevos procesos pueden ser independientes y no compartir el espacio de memoria con el proceso que los ha creado o ser creados en el mismo espacio de memoria.

Estados y transiciones de los procesos

El principal trabajo del procesador es ejecutar las instrucciones de máquina que se encuentran en memoria principal. Estas instrucciones se encuentran en forma de programas. Para que un programa pueda ser ejecutado, el sistema operativo crea un nuevo proceso, y el procesador ejecuta una tras otra las instrucciones del mismo.

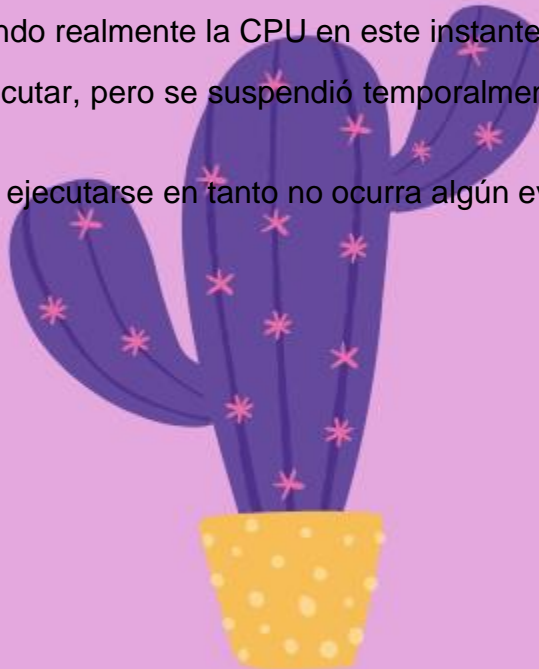
En un entorno de multiprogramación, el procesador intercalará la ejecución de instrucciones de varios programas que se encuentran en memoria. El sistema operativo es el responsable de determinar las pautas de intercalado y asignación de recursos a cada proceso. Aunque cada proceso es una entidad independiente, con su propio contador de programa y estado interno, los procesos a menudo necesitan

interactuar con otros procesos. Un proceso podría generar ciertas salidas que otro proceso utilizan como entradas, en el comando de Shell.

Cuando un proceso se bloquea, lo que hace porque le es imposible continuar lógicamente, casi siempre porque está separando entradas que todavía no están disponibles, también puede ser que un programa que conceptualmente está listo y en condiciones de ejecutarse sea detenido porque el sistema operativo ha decidido asignar la CPU a otro proceso durante un tiempo.

Estas dos condiciones son totalmente distintas, en el primer caso, la suspensión es inherente al problema (no es posible procesar la línea de comandos del usuarios antes de que este la teclee). En el segundo caso, se trata de un tecnicismo del sistema (no hay suficiente: CPU para darle a cada proceso su propio procesador privado).

- 1.- Ejecutándose (usando realmente la CPU en este instante).
- 2.- Listo (se puede ejecutar, pero se suspendió temporalmente para dejar que otro proceso se ejecute).
- 3.- Bloqueo (no puede ejecutarse en tanto no ocurra algún evento externo).



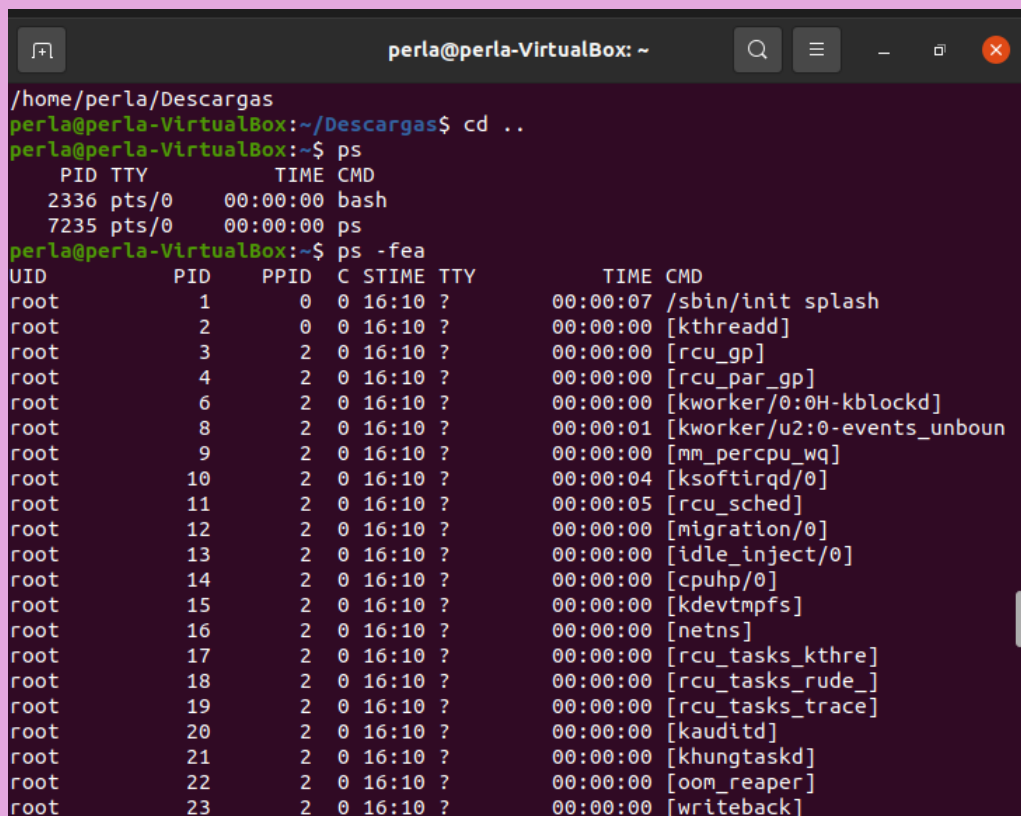
DESARROLLO.

Sección Linux:

1. Introduzca los siguientes comandos a través de la consola del sistema operativo Linux: `ps` `ps -fea` ¿Qué información le proporcionan los comandos anteriores?

El comando `ps` muestra por pantalla un listado de los procesos que se están ejecutando en el sistema.

PID : Id del Proceso, **TTY** : Terminal., **TIME** : Tiempo de ejecución, **CMD** : Comando.



```
perla@perla-VirtualBox: ~  
/home/perla/Descargas  
perla@perla-VirtualBox:~/Descargas$ cd ..  
perla@perla-VirtualBox:~$ ps  
  PID TTY          TIME CMD  
 2336 pts/0    00:00:00 bash  
 7235 pts/0    00:00:00 ps  
perla@perla-VirtualBox:~$ ps -fea  
UID          PID    PPID  C  STIME TTY          TIME CMD  
root           1         0  0  16:10 ?           00:00:07 /sbin/init splash  
root           2         0  0  16:10 ?           00:00:00 [kthreadd]  
root           3         2  0  16:10 ?           00:00:00 [rcu_gp]  
root           4         2  0  16:10 ?           00:00:00 [rcu_par_gp]  
root           6         2  0  16:10 ?           00:00:00 [kworker/0:0H-kblockd]  
root           8         2  0  16:10 ?           00:00:01 [kworker/u2:0-events_unboun  
root           9         2  0  16:10 ?           00:00:00 [mm_percpu_wq]  
root          10         2  0  16:10 ?           00:00:04 [ksoftirqd/0]  
root          11         2  0  16:10 ?           00:00:05 [rcu_sched]  
root          12         2  0  16:10 ?           00:00:00 [migration/0]  
root          13         2  0  16:10 ?           00:00:00 [idle_inject/0]  
root          14         2  0  16:10 ?           00:00:00 [cpuhp/0]  
root          15         2  0  16:10 ?           00:00:00 [kdevtmpfs]  
root          16         2  0  16:10 ?           00:00:00 [netns]  
root          17         2  0  16:10 ?           00:00:00 [rcu_tasks_kthre]  
root          18         2  0  16:10 ?           00:00:00 [rcu_tasks_rude_  
root          19         2  0  16:10 ?           00:00:00 [rcu_tasks_trace]  
root          20         2  0  16:10 ?           00:00:00 [kauditd]  
root          21         2  0  16:10 ?           00:00:00 [khungtaskd]  
root          22         2  0  16:10 ?           00:00:00 [oom_reaper]  
root          23         2  0  16:10 ?           00:00:00 [writeback]
```

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.

1.1 Fork()

Mediante la llamada al sistema fork() se crean nuevos procesos en el sistema.. Un proceso que hace una llamada a fork() hará que el sistema cree una copia del proceso original (llamado entonces padre) para que los dos procesos sigan sus caminos independientemente. El nuevo proceso, llamado hijo, tiene un nuevo PID. El PPID (parent's PID) será el del proceso que ha llamado a fork(): el proceso padre. Para que los procesos sepan diferenciar quien es proceso padre y quien el hijo (y puedan seguir sus caminos) al código de retorno de fork() es diferente del padre que del hijo. En el caso del proceso padre, se le devuelve:

- PID del proceso hijo si se ha creado
- 1 si no se ha podido crear el proceso hijo

El proceso hijo, siempre se le devolverá el valor 0, por lo que sabrá que es el hijo

1.2 getpid()

Devuelve el ID de proceso del proceso actual. (Esto a menudo lo utilizan las rutinas que generan nombres de archivo temporales únicos).

1.3 getppid()

Devuelve el ID de proceso del padre del proceso actual.

obtener identificación del proceso

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

pid_t getpid(vacío);
pid_t getppid (vacío);
```

1.4 Wait()

pid_t wait(int *status)

Permite a un proceso padre esperar hasta que termine un proceso hijo.

El entero apuntado por el argumento status será actualizado con un código que indica el estado de terminación del proceso hijo.

Devuelve el identificador del proceso hijo ó -1 en caso de error

1.5 execv()

La función `execv()` reemplaza la imagen de proceso actual con una nueva imagen de proceso especificada por *ruta*. La nueva imagen se construye a partir de un archivo ejecutable normal llamado nuevo archivo de imagen de proceso. No se realiza ninguna devolución porque la imagen del proceso que llama se reemplaza por la nueva imagen del proceso.

1.6 Funciones similares a `execv()`

La familia `exec...()` es un conjunto de funciones que en esencia realizan la misma actividad ya que solo difieren en la forma de pasar sus argumentos, son utilizadas para poner en ejecución un proceso determinado, la característica es que las instrucciones del proceso que las invoca son sustituidas por las instrucciones del proceso indicado. Dichas funciones son:

`execl, execlp, execl, execv, execvp`
`execl(const char *path, const char *arg, ...);`
`execlp(const char *file, const char *arg, ...);`
`execl(const char *path, const char *arg, ..., char *const envp[]);`
`execv(const char *path, char *const argv[]);`
`execvp(const char *file, char *const argv[]);`

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);
}
```



```
perla@perla-VirtualBox:~/Descargas$ gcc Programa1.c -o Programa1
perla@perla-VirtualBox:~/Descargas$ ./Programa1
Soy el proceso padre
perla@perla-VirtualBox:~/Descargas$ Soy el proceso hijo

perla@perla-VirtualBox:~/Descargas$ ./Programa1
Soy el proceso padre
perla@perla-VirtualBox:~/Descargas$ Soy el proceso hijo
```

```
perla@perla-VirtualBox:~/Descargas$ gcc Programa2.c -o Programa2
perla@perla-VirtualBox:~/Descargas$ ./Programa2
Soy el proceso padre
Mensaje en ambos
perla@perla-VirtualBox:~/Descargas$ Soy el proceso hijo
Mensaje en ambos
```

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 int main(void)
5 {
6     int id_proc;
7     id_proc=fork();
8     if (id_proc == 1)
9     {
10        printf("Soy el proceso hijo\n");
11    }
12    else
13    {
14        printf("Soy el proceso padre\n");
15    }
16    printf("Mensaje en ambos\n");
17    exit(0);
18 }
```

```
perla@perla-VirtualBox:~/Descargas$ gcc Programa2.c -o Programa2
perla@perla-VirtualBox:~/Descargas$ ./Programa2
Soy el proceso padre
Mensaje en ambos
perla@perla-VirtualBox:~/Descargas$ Soy el proceso padre
Mensaje en 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 (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

```
GNU nano 4.8 arbol.c
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/wait.h>

int main(void) {
    int i,j,k,id,id_H;
    for(i=0; i<5; i++){
        id = fork();
        wait(NULL);
        if(id == 0) {
            printf("Proceso padre: %d /t Proceso hijo %d \n\n", getppid(), getpid());

            if(i == 0){
                for(j=0; j<5; j++){
                    id_H = fork();
                    if(id_H !=0) {
                        wait(NULL);
                        printf("Proceso padre: %d /t Proceso hijo %d \n\n", getppid(), getpid());
                        break;
                    }
                }
            }

            if(i == 1) {
                for(j=0; j<4; j++){
                    id_H = fork();
                    if(id_H !=0){
                        wait(NULL);
                        printf("Proceso padre: %d /t Proceso hijo %d \n\n", getppid(), getpid());
                        break;
                    }
                }
            }
        }
    }
}
```

```
GNU nano 4.8 arbol.c
        if(i == 2) {
            for(j=0; j<3; j++){
                id_H = fork();
                if(id_H !=0){
                    wait(NULL);
                    printf("Proceso padre: %d /t Proceso hijo %d \n\n", getppid(), getpid());
                    break;
                }
            }

            if(i == 3) {
                for(j=0; j<2; j++){
                    id_H = fork();
                    if(id_H !=0){
                        wait(NULL);
                        printf("Proceso padre: %d /t Proceso hijo %d \n\n", getppid(), getpid());
                        break;
                    }
                }
            }

            if(i == 4) {
                for(j=0; j<1; j++){
                    id_H = fork();
                    if(id_H !=0){
                        wait(NULL);
                        printf("Proceso padre: %d /t Proceso hijo %d \n\n", getppid(), getpid());
                        break;
                    }
                }
            }
        }
    }
}
```

Procesos impresos:

```
Proceso padre: 1933 /t Proceso hijo 10730
Proceso padre: 1933 /t Proceso hijo 10731
Proceso padre: 1933 /t Proceso hijo 10732
Proceso padre: 1933 /t Proceso hijo 10733
Proceso padre: 1933 /t Proceso hijo 10734
Proceso padre: 1933 /t Proceso hijo 10735
Proceso padre: 1933 /t Proceso hijo 10736
Proceso padre: 1933 /t Proceso hijo 10737
Proceso padre: 1933 /t Proceso hijo 10738
Proceso padre: 1933 /t Proceso hijo 10739
Proceso padre: 1933 /t Proceso hijo 10740
Proceso padre: 1933 /t Proceso hijo 10741
Proceso padre: 1933 /t Proceso hijo 10742
Proceso padre: 1933 /t Proceso hijo 10743
Proceso padre: 1933 /t Proceso hijo 10744
Proceso padre: 1933 /t Proceso hijo 10745
Proceso padre: 1933 /t Proceso hijo 10746
Proceso padre: 1933 /t Proceso hijo 10747
Proceso padre: 1933 /t Proceso hijo 10748
Proceso padre: 1933 /t Proceso hijo 10749
```

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.

IMPRESIÓN:

```
perla@perla-VirtualBox:~/Escritorio/PRACTICA 2$ ./matrices_Proceso

Resultado del la Suma

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

Resultado del la Resta

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
0 0 0 0 0 0 0
2
2 4 6 8 10 12 14

Resultado del la multiplicacion

1 4 9 16 25 36 49
49 36 25 16 9 4 1
9 4 1 16 25 36 49
49 36 25 1 4 9 16
36 9 25 4 16 1 49
49 16 25 1 4 9 36
1 4 9 16 25 36 49

Resultado del la Transpuesta

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

1 7 3 7 6 7 1
2 6 2 6 3 4 2
3 5 1 5 5 5 3
4 4 4 1 2 1 4
5 3 5 2 4 2 5
6 2 6 3 1 3 6
7 1 7 4 7 6 7
perla@perla-VirtualBox:~/Escritorio/PRACTICA 2$
```

Programa sin procesos

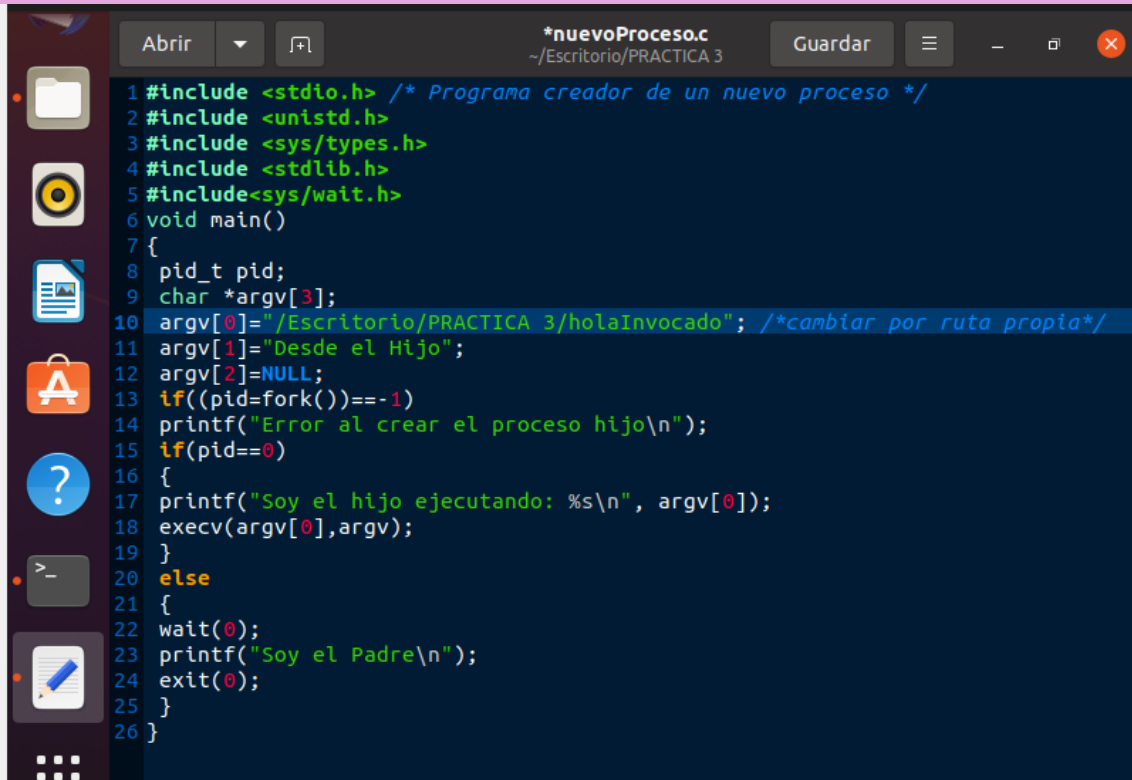
```
real    0m0.186s
user    0m0.002s
sys     0m0.000s
```

Programa con Procesos

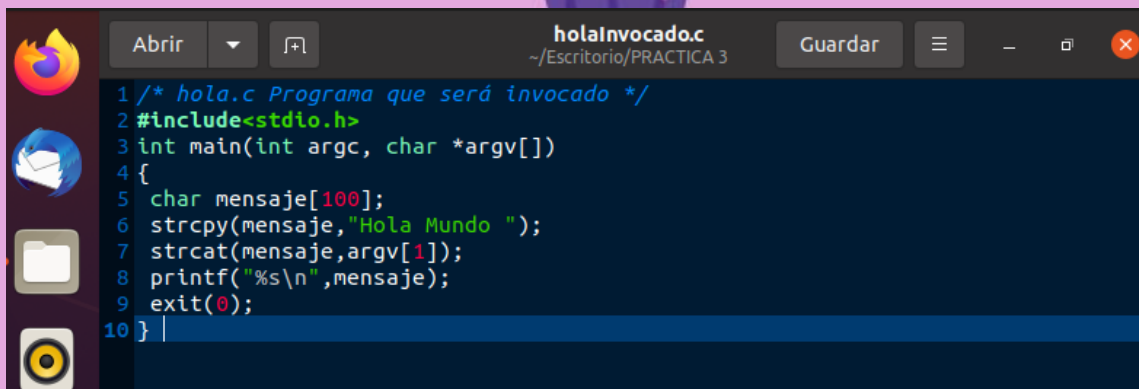
```
real    0m0.004s
user    0m0.002s
... sys  0m0.002s
```

se reduce mucho el tiempo de ejecución cuando empleamos procesos, y que sin procesos puede tardar .186 segundos y con proceso

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.

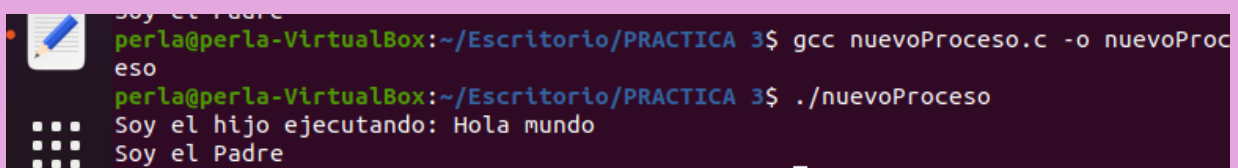


```
1#include <stdio.h> /* Programa creador de un nuevo proceso */
2#include <unistd.h>
3#include <sys/types.h>
4#include <stdlib.h>
5#include <sys/wait.h>
6void main()
7{
8    pid_t pid;
9    char *argv[3];
10   argv[0]="/Escritorio/PRACTICA 3/holaInvocado"; /*cambiar por ruta propia*/
11   argv[1]="Desde el Hijo";
12   argv[2]=NULL;
13   if((pid=fork())==-1)
14   printf("Error al crear el proceso hijo\n");
15   if(pid==0)
16   {
17   printf("Soy el hijo ejecutando: %s\n", argv[0]);
18   execv(argv[0],argv);
19   }
20   else
21   {
22   wait(0);
23   printf("Soy el Padre\n");
24   exit(0);
25   }
26 }
```



```
1/* hola.c Programa que será invocado */
2#include<stdio.h>
3int main(int argc, char *argv[])
4{
5    char mensaje[100];
6    strcpy(mensaje,"Hola Mundo ");
7    strcat(mensaje,argv[1]);
8    printf("%s\n",mensaje);
9    exit(0);
10 }
```

IMPRIME:



```
perla@perla-VirtualBox:~/Escritorio/PRACTICA 3$ gcc nuevoProceso.c -o nuevoProceso
perla@perla-VirtualBox:~/Escritorio/PRACTICA 3$ ./nuevoProceso
Soy el hijo ejecutando: Hola mundo
Soy el Padre
```

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.

```
FORMULA GENERAL (PROGRAMA):
El resultado 1(+) es: -1.500000 , el resultado 2(-) es: -1.500000
PROGRAMA FIBONACCI
ingrese la cantidad de numeros 8
1, 1 ,
2 ,
3 ,
5 ,
8 ,
13 ,
21 ,

PROGRAMA MULTIPLICACION
1 ,4 ,9 ,16 ,25 ,36 ,49 ,49 ,36 ,25 ,16 ,9 ,4 ,1 ,9 ,4 ,1 ,16 ,25 ,36 ,49 ,49 ,
36 ,25 ,1 ,4 ,9 ,16 ,36 ,9 ,25 ,4 ,16 ,1 ,49 ,49 ,16 ,25 ,1 ,4 ,9 ,36 ,1 ,4 ,9 ,
16 ,25 ,36 ,49 ,perla@perla-VirtualBox:~/Escritorio/PRACTICA 3$
```

CODIGO

```
F.General,Fibonacci,multiplicacion.c
~/Escritorio/PRACTICA 3

1#include<stdio.h>
2#include<unistd.h>
3#include<stdlib.h>
4#include<sys/wait.h>
5#include<fcntl.h>
6#include<string.h>
7#include<math.h>
8
9void main() {
10    int i,j,k,id;
11    int matriz1[7][7] = {{1,2,3,4,5,6,7},{7,6,5,4,3,2,1},-
{3,2,1,4,5,6,7},{7,6,5,1,2,3,4},{6,3,5,2,4,1,7},{7,4,5,1,2,3,6},-
{1,2,3,4,5,6,7}};
12    int matriz2[7][7] = {{1,2,3,4,5,6,7},{7,6,5,4,3,2,1},-
{3,2,1,4,5,6,7},{7,6,5,1,2,3,4},{6,3,5,2,4,1,7},{7,4,5,1,2,3,6},-
{1,2,3,4,5,6,7}};
13    double a=2,b=6,c=2,cuadrado,resta,div,neg,n,raiz=0,res1,res2,numero;
14    int a1,b1,n1,c1,l;
15
16    for(i=0; i<3; i++){
17        id = fork();
18        wait(NULL);
19        if(id == 0){
20            if(i == 0){
21                printf("\n\n FORMULA GENERAL (PROGRAMA): \n");
22
23                neg = b*(-1.0);
```

```
F.General, fibonacci, multiplicacion.c
~/Escritorio/PRACTICA 3

24         cuadrado = b*b;
25         resta = (4.0)*(a*c);
26         numero = (cuadrado - resta);
27         //raiz = sqrt (numero);
28         res1= (neg+raiz)/(2.0*a);
29         res2= (neg - raiz)/ (2.0*a);
30         printf("El resultado 1(+) es: %f , el resultado
31         2(-) es: %f", res1, res2);
32     }
33     if(i == 1) {
34         printf("\n PROGRAMA FIBONACCI \n");
35         a1 = 0; b1 = 1; c1 = 1; //n1=10;
36         printf ("ingrese la cantidad de numeros");
37         scanf ("%i",&n1);
38         printf ("1, ");
39         for(j=0; j<n1-1; j++){
40             c1 = a1 + b1;
41             a1 = b1;
42             b1 = c1;
43             printf("%d ,", c1);
44             printf("\n\n");
45         }
46         if (i==2){
47             printf("\n PROGRAMA MULTIPLICACION \n");
48             for(k=0; k<7; k++){
49                 for(j=0; j<7; j++){
50                     l = (matriz2[k][j] * matriz1[k][j]);
51                     printf("%d ,", l);
52                 }
53             }
54             break;
55         }
56     }
57 }
58 }
59
60
```

Podemos decir que los tres programas han sido ejecutados en diferentes subprocesos.

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.

CÓDIGO:

```
matrices_Proceso.c
~/Escritorio/PRACTICA 2

1 #include<stdio.h>
2 #include<unistd.h>
3 #include<stdlib.h>
4 #include<sys/wait.h>
5 #include<fcntl.h>
6 #include<string.h>
7
8 int main(void){
9     int File1;
10    int i,j,k,id;
11    int matriz1[7][7] = {{1,2,3,4,5,6,7},{7,6,5,4,3,2,1},-
12    {3,2,1,4,5,6,7},{7,6,5,1,2,3,4},{6,3,5,2,4,1,7},{7,4,5,1,2,3,6},-
13    {1,2,3,4,5,6,7}};
14    int matriz2[7][7] = {{1,2,3,4,5,6,7},{7,6,5,4,3,2,1},-
15    {3,2,1,4,5,6,7},{7,6,5,1,2,3,4},{6,3,5,2,4,1,7},{7,4,5,1,2,3,6},-
16    {1,2,3,4,5,6,7}};
17    int ans[7][7],cont=0, l;
18    char ll[100];
19    char answer[400];
20    ssize_t nr_bytes;
21
22    for(i=0; i<5; i++){
23        id = fork();
24        wait(NULL);
25        if(id == 0){
26            //printf("Proceso padre: %d Proceso hijo:
27            %d\n\n",getppid(), getpid());
28
29            if(i == 0){
30                for(k = 0; k<7; k++){
31                    for(j=0; j<7;j++){
32                        l = matriz2[k][j] +
33                        matriz1[k][j];
34
35                        sprintf(ll, "%d", l);
36                        strcat(answer, ll);
37                        strcat(answer, " ");
38
39                        if(l>=0 && l<10){cont++;}
40                        if(l>=10 && l<100){cont+=2;}
41                        if(j != 6){cont++;}
42                    }
43                    strcat(answer, "\n"); cont+=2;
44                }
45
46                File1 =
47                open("Suma.txt",O_CREAT|O_WRONLY,0700);
48                write(File1,answer,cont);
49                close(File1);
50            }
51
52            if(i == 1){
53                for(k = 0; k<7; k++){
54                    for(i=0; i<7;i++){
```



```

39         write(File1,answer,cont);
40         close(File1);
41     }
42
43     if(i == 1){
44         for(k = 0; k<7; k++){
45             for(j=0; j<7;j++){
46                 l = matriz2[k][j] -
matriz1[k][j];
47
48                 sprintf(ll, "%d", l);
49                 strcat(answer, ll);
50                 strcat(answer, " ");
51
52                 if(l>=0 && l<10){cont++;}
53                 if(l>=10 && l<100){cont+=2;}
54                 if(l<0 && l>-10){cont+=2;}
55                 if(l<=-10 && l>-100)
{cont+=3;}
56
57                 if(j != 6){cont++;}
58             }
59             strcat(answer, "\n"); cont+=2;
60         }
61
62         File1 =
open("Resta.txt",O_CREAT|O_WRONLY,0600);
63         write(File1,answer,cont);
        close(File1);

```

```

64     }
65
66     if(i == 2) {
67         for(k = 0; k<7; k++){
68             for(j=0; j<7;j++){
69                 l = matriz2[k][j] *
matriz1[k][j];
70
71                 sprintf(ll, "%d", l);
72                 strcat(answer, ll);
73                 strcat(answer, " ");
74
75                 if(l>=0 && l<10){cont++;}
76                 if(l>=10 && l<100){cont+=2;}
77                 if(l<0 && l>-10){cont+=2;}
78                 if(l<=-10 && l>-100)
{cont+=3;}
79
80                 if(j != 6){cont++;}
81             }
82             strcat(answer, "\n");
83             cont+=2;
84         }
85
86         File1 =
open("Multiplicacion.txt",O_CREAT|O_WRONLY,0700);
87         write(File1,answer,cont);
        close(File1);

```

```

85         close(File1);
86     }
87
88     if(i == 3){
89         for(k = 0; k<7; k++){
90             for(j=0; j<7;j++){
91                 l = matriz2[j][k];
92
93                 sprintf(ll, "%d", l);
94                 strcat(answer, ll);
95                 strcat(answer, " ");
96
97                 if(l>=0 && l<10){cont++;}
98                 if(l>=10 && l<100){cont+=2;}
99                 if(l<0 && l>-10){cont+=2;}
100                 if(l<=-10 && l>-100)
101 {cont+=3;}
102
103             }
104             strcat(answer, "\n");
105             cont+=2;
106             strcat(answer, "\n");
107             for(k = 0; k<7; k++){
108                 for(j=0; j<7;j++){

```

```

104             }
105             strcat(answer, "\n");
106             cont+=2;
107             strcat(answer, "\n");
108             for(k = 0; k<7; k++){
109                 for(j=0; j<7;j++){
110                     l = matriz1[j][k];
111
112                     sprintf(ll, "%d", l);
113                     strcat(answer, ll);
114                     strcat(answer, " ");
115
116                     if(l>=0 && l<10){cont++;}
117                     if(l>=10 && l<100){cont+=2;}
118                     if(l<0 && l>-10){cont+=2;}
119                     if(l<=-10 && l>-100)
120 {cont+=3;}
121
122                 }
123                 strcat(answer, "\n");
124                 cont+=2;
125                 }
126                 File1 = open("Transpuesta.txt",O_CREAT|-
127                 O_WRONLY,0700);
128                 write(File1,answer,cont);
129                 close(File1);

```

```

119         if(j != 0){cont++;}
120     }
121     strcat(answer, "\n");
122     cont+=2;
123 }
124 File1 = open("Transpuesta.txt",O_CREAT|-
O_WRONLY,0700);
125 write(File1,answer,cont);
126 close(File1);
127 }
128
129 if(i == 4) {
130     printf("\nResultado del la Suma \n\n");
131     File1 = open("Suma.txt",O_RDONLY);
132     nr_bytes = read(File1, answer, 200);
133     printf("%s", answer);
134     close(File1);
135
136     printf("\nResultado del la Resta\n\n");
137     File1 = open("Resta.txt",O_RDONLY);
138     nr_bytes = read(File1, answer, 200);
139     printf("%s", answer);
140     close(File1);
141
142     printf("\nResultado del la
multiplicacion\n\n");
143     File1 = open("Multiplicacion.txt",O_RDONLY);
144     nr_bytes = read(File1, answer, 200);
145     printf("%s", answer);
146     close(File1);
147
148     printf("\nResultado del la
multiplicacion\n\n");
149     File1 = open("Multiplicacion.txt",O_RDONLY);
150     nr_bytes = read(File1, answer, 200);
151     printf("%s", answer);
152     close(File1);
153 }
154 break;
155 }
156 }
157 }

```

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);

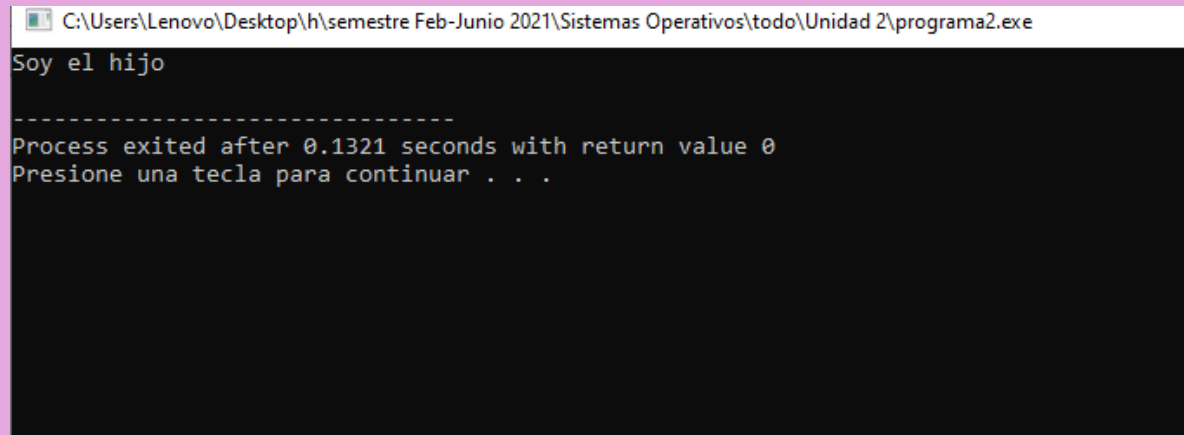
    // Terminación controlada del proceso e hilo asociado de ejecución
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

```
C:\Users\Lenovo\Desktop\h\semestre Feb-Junio 2021\Sistemas Operativos\todo\Unidad 2\programa2.exe
Usar: C:\Users\Lenovo\Desktop\h\semestre Feb-Junio 2021\Sistemas Operativos\todo\Unidad 2\programa2.exe Nombre_programa_hijo
-----
Process exited after 0.1199 seconds with return value 0
Presione una tecla para continuar . . .
```

4. Capture y compile el programa que contendrá al proceso hijo que a continuación se muestra.

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

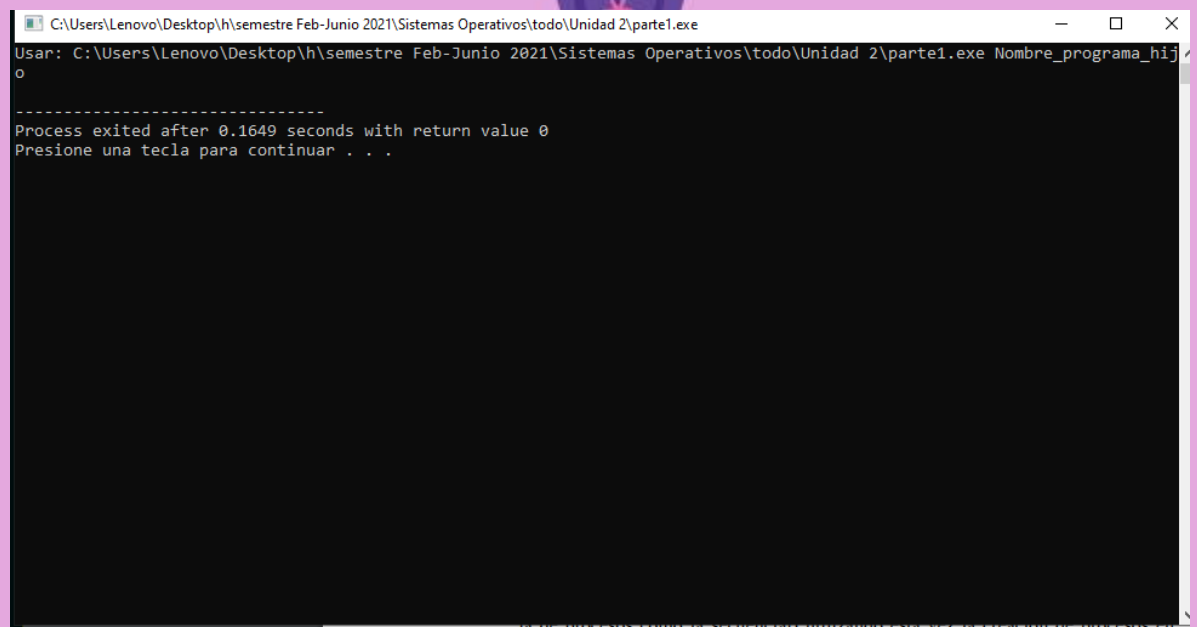
int main(void)
{
    printf("Soy el hijo\n");
    exit(0);
}
```



C:\Users\Lenovo\Desktop\h\semestre Feb-Junio 2021\Sistemas Operativos\todo\Unidad 2\programa2.exe

```
Soy el hijo
-----
Process exited after 0.1321 seconds with return value 0
Presione una tecla para continuar . . .
```

5.-Ejecute el primer código pasando como argumento el nombre del archivo ejecutable del segundo código capturado. Observe el funcionamiento del programa, reporte sus observaciones y experimente con el código



C:\Users\Lenovo\Desktop\h\semestre Feb-Junio 2021\Sistemas Operativos\todo\Unidad 2\parte1.exe

```
Usar: C:\Users\Lenovo\Desktop\h\semestre Feb-Junio 2021\Sistemas Operativos\todo\Unidad 2\parte1.exe Nombre_programa_hijo
-----
Process exited after 0.1649 seconds with return value 0
Presione una tecla para continuar . . .
```

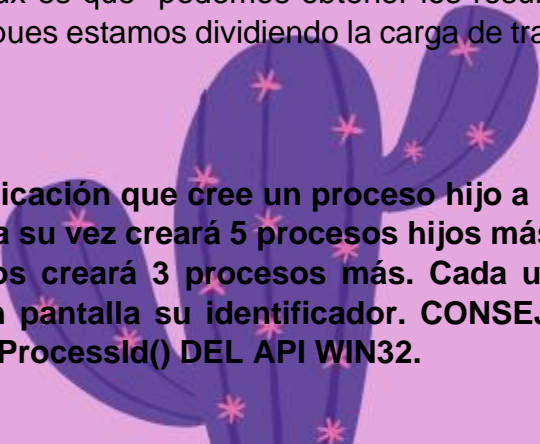
Solo se imprimió una O lo que significa que no tomo los argumentos del otro programa correctamente

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

Cabe destacar la comprensión del concepto fork, que es como la acción de crear una copia exacta del programa principal, y en el transcurso de la práctica se crearon varias copias que desarrollaran un trabajo en específico Lo interesante de todo esto es el hecho de tener acceso a algunas partes importantes del sistema operativo, y observar el funcionamiento interno del mismo, y poder aprovechar la capacidad multitarea de Linux.

Una de las similitudes al utilizar la implementación de multiprocesos tanto en Windows como en Linux es que podemos obtener los resultados de una manera más rápida y eficiente pues estamos dividiendo la carga de trabajo en procesos que se ejecutan a la vez.

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.



```
1  #include <stdio.h>
2  #include <windows.h>
3  |
4  int main(){
5      STARTUPINFO si;
6      PROCESS_INFORMATION pi;
7      ZeroMemory(&(si),sizeof(si));
8      si.cb=sizeof(si);
9      ZeroMemory(&(pi),sizeof(pi));
10     printf("ID Padre: %d\n",GetCurrentProcessId());
11     for(int i=0;i<5;i++){
12         if(!CreateProcess(NULL,"PrimerNivel",NULL,NULL,FALSE,0,NULL,NULL,&(si),&(pi)))
13         {
14             printf("fallo al invocar CreateProcess (%d)\n",GetLastError());
15             return 1;
16         }
17         WaitForSingleObject(pi.hProcess,INFINITE);
18         CloseHandle(pi.hProcess);
19         CloseHandle(pi.hThread);
20     }
21     return 0;
22 }
```

Creamos al padre el cual contendrá los 5 hijos

```

1 #include <stdio.h>
2 #include <windows.h>
3 int main(){
4     STARTUPINFO si;
5     PROCESS_INFORMATION pi;
6     ZeroMemory(&si,sizeof(si));
7     si.cb=sizeof(si);
8     ZeroMemory(&pi,sizeof(pi));
9     printf("\tID Hijo Primer Nivel: %d\n",GetCurrentProcessId());
10    for(int i=0;i<3;i++){
11        if(!CreateProcess(NULL,"SegundoNivel",NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){
12            printf("fallo al invocar CreateProcess (%d)\n",GetLastError());
13            return 1;
14        }
15        WaitForSingleObject(pi.hProcess,INFINITE);
16        CloseHandle(pi.hProcess);
17        CloseHandle(pi.hThread);
18    }
19    return 0;
20 }

```

Construimos el segundo nivel el cual contendrá a los 3 hijos restantes los cuales serán llamados

```

1 #include <stdio.h>
2 #include <windows.h>
3 int main(){
4     printf("\t\tID Hijo Segundo Nivel: %d\n",GetCurrentProcessId());
5     return 0;
6 }

```

```

C:\Users\Lenovo\Desktop\h\semestre Feb-Junio 2021\Sistemas Operativos\todo\Unidad 2\padre.exe
ID Padre: 13128
  ID Hijo Primer Nivel: 11396
    ID Hijo Segundo Nivel: 7356
    ID Hijo Segundo Nivel: 3264
    ID Hijo Segundo Nivel: 6004
  ID Hijo Primer Nivel: 7856
    ID Hijo Segundo Nivel: 14344
    ID Hijo Segundo Nivel: 13544
    ID Hijo Segundo Nivel: 14756
  ID Hijo Primer Nivel: 704
    ID Hijo Segundo Nivel: 3372
    ID Hijo Segundo Nivel: 6948
    ID Hijo Segundo Nivel: 9968
  ID Hijo Primer Nivel: 5456
    ID Hijo Segundo Nivel: 9784
    ID Hijo Segundo Nivel: 14076
    ID Hijo Segundo Nivel: 6376
  ID Hijo Primer Nivel: 12152
    ID Hijo Segundo Nivel: 9272
    ID Hijo Segundo Nivel: 4200
    ID Hijo Segundo Nivel: 12784

-----
Process exited after 0.6521 seconds with return value 0
Presione una tecla para continuar . . .

```

Aquí podemos ver como el proceso hijo creado a partir de una función padre crea a los 5 hijos los cuales llaman a otros 3 procesos hijos

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  #include <stdio.h>
2  #include <unistd.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  float mat1[40][40]={1,0,0,0,0,0,0,0,0,0},
7                      {1,2,0,0,0,0,0,0,0,0},
8                      {1,2,3,0,0,0,0,0,0,0},
9                      {1,2,3,4,0,0,0,0,0,0},
10                     {1,2,3,4,5,0,0,0,0,0},
11                     {1,2,3,4,5,6,0,0,0,0},
12                     {1,2,3,4,5,6,7,0,0,0},
13                     {1,2,3,4,5,6,7,8,0,0},
14                     {1,2,3,4,5,6,7,8,9,0},
15                     {1,2,3,4,5,6,7,8,9,10}};
16
17 float mat2[40][40]={10,9,8,7,6,5,4,3,2,1},
18                     {0,9,8,7,6,5,4,3,2,1},
19                     {0,0,8,7,6,5,4,3,2,1},
20                     {0,0,0,7,6,5,4,3,2,1},
21                     {0,0,0,0,6,5,4,3,2,1},
22                     {0,0,0,0,0,5,4,3,2,1},
23                     {0,0,0,0,0,0,4,3,2,1},
```

```

25         {0,0,0,0,0,0,0,0,2,1},
26         {0,0,0,0,0,0,0,0,0,1}};
27 int mat1t[10][10]={};
28 int mat2t[10][10]={};
29 int mat1inv[10][10]={};
30 int mat2inv[10][10]={};
31 int multi[10][10]={};
32
33 char archivo[10000];
34
35
36 void InverseOfMatrix(float matrix[][40], int order,int opc){
37
38     float temp;
39     for (int i = 0; i < order; i++) {
40         for (int j = 0; j < 2 * order; j++) {
41             if (j == (i + order))
42                 matrix[i][j] = 1;
43         }
44     }
45
46     for (int i = order - 1; i > 0; i--) {
47         if (matrix[i - 1][0] < matrix[i][0])
48             for (int j = 0; j < 2 * order; j++) {
49                 temp = matrix[i][j];
50                 matrix[i][j] = matrix[i - 1][j];
51                 matrix[i - 1][j] = temp;
52             }
53     }
54
55     for (int i = 0; i < order; i++) {
56         for (int j = 0; j < 2 * order; j++) {
57             if (j != i) {
58                 temp = matrix[j][i] / matrix[i][i];
59                 for (int k = 0; k < 2 * order; k++) {
60                     matrix[j][k] -= matrix[i][k] * temp;
61                 }
62             }
63         }
64     }
65
66     for (int i = 0; i < order; i++) {
67
68         temp = matrix[i][i];
69         for (int j = 0; j < 2 * order; j++) {
70

```

```

71         matrix[i][j] = matrix[i][j] / temp;
72     }
73 }
74
75 for (int i = 0; i < order; i++) {
76     for (int j = order; j < 2 * order; j++) {
77         if(opc==0)
78             mat1inv[i][j-order]=matrix[i][j];
79         else
80             mat2inv[i][j-order]=matrix[i][j];
81     }
82 }
83
84 return;
85 }
86
87 void leer(char entrada[]){
88     char *secuencia;
89     FILE *ptrs;
90     ptrs= fopen(entrada,"r");
91     if(ptrs==NULL) {
92         printf("No hay datos");
93         exit(1);
94     }else{
95         while (fgets((char*)&archivo, sizeof(archivo), ptrs)) {
96             printf("%s",archivo);
97         }
98         fclose(ptrs);
99     }
100 }
101
102 int main(void){
103     double total_time;
104     clock_t start, end;
105     start = clock();
106     FILE* fichero;
107     fichero = fopen("suma.txt", "w");
108     for(int i=0; i<10; i++) {
109         for(int j=0; j<10; j++) {
110             //printf("%d ",(int)(mat1[i][j]+mat2[i][j]));
111             fprintf(fichero, "%d ", (int)(mat1[i][j]+mat2[i][j]));
112         }
113         fprintf(fichero, "\n");
114     }
115     fclose(fichero);

```

```

116
117 fichero = fopen("resta.txt", "w");
118 for(int i=0; i<10; i++) {
119     for(int j=0; j<10; j++) {
120         //printf("%d ", (int)(mat1[i][j]-mat2[i][j]));
121         fprintf(fichero, "%d ", (int)(mat1[i][j]-mat2[i][j]));
122     }
123     fprintf(fichero, "\n");
124 }
125 fclose(fichero);
126
127 fichero = fopen("multi.txt", "w");
128
129 int i, j, k;
130 for (i = 0; i < 10; i++) {
131     for (j = 0; j < 10; j++) {
132         multi[i][j] = 0;
133         for (k = 0; k < 10; k++)
134             multi[i][j] += mat1[i][k]*mat2[k][j];
135     }
136 }
137
138 for(int i=0; i<10; i++) {

```

```

138     for(int i=0; i<10; i++) {
139         for(int j=0; j<10; j++) {
140             fprintf(fichero, "%d ", (int)(multi[i][j]));
141         }
142         fprintf(fichero, "\n");
143     }
144
145     for(int i=0; i<10; i++) {
146         for(int j=0; j<10; j++) {
147             mat1t[i][j] = mat1[j][i];
148             mat2t[i][j] = mat2[j][i];
149         }
150     }
151
152     fichero = fopen("traspuesta.txt", "w");
153     for(int i=0; i<10; i++) {
154         for(int j=0; j<10; j++) {
155             fprintf(fichero, "%d ", (int)(mat1t[i][j]));
156         }
157         fprintf(fichero, "\n");
158     }
159     fprintf(fichero, "\n");
160     for(int i=0; i<10; i++) {

```

```

178     for(int i=0; i<10; i++) {
179         for(int j=0; j<10; j++) {
180             fprintf(fichero, "%.3f ", (float)(mat2inv[i][j]));
181         }
182         fprintf(fichero, "\n");
183     }
184     fclose(fichero);
185
186     printf("\n=== Suma ===\n");
187     leer("suma.txt");
188     printf("\n=== Resta ===\n");
189     leer("resta.txt");
190     printf("\n=== Multiplicacion ===\n");
191     leer("multi.txt");
192     printf("\n=== Traspuesta ===\n");
193     leer("traspuesta.txt");
194     printf("\n=== Inversa ===\n");
195     leer("inversa.txt");
196     end = clock();
197     total_time = ((double) (end - start)) / CLK_TCK;
198     printf("\n\t---El tiempo de ejecucion fue de: %f\n", total_time);
199     return 0;
200 }

```

```

=== Suma ===
11 9 8 7 6 5 4 3 2 1
1 11 8 7 6 5 4 3 2 1
1 2 11 7 6 5 4 3 2 1
1 2 3 11 6 5 4 3 2 1
1 2 3 4 11 5 4 3 2 1
1 2 3 4 5 11 4 3 2 1
1 2 3 4 5 6 11 3 2 1
1 2 3 4 5 6 7 11 2 1
1 2 3 4 5 6 7 8 11 1
1 2 3 4 5 6 7 8 9 11

=== Resta ===
-9 -9 -8 -7 -6 -5 -4 -3 -2 -1
1 -7 -8 -7 -6 -5 -4 -3 -2 -1
1 2 -5 -7 -6 -5 -4 -3 -2 -1
1 2 3 -3 -6 -5 -4 -3 -2 -1
1 2 3 4 -1 -5 -4 -3 -2 -1
1 2 3 4 5 1 -4 -3 -2 -1
1 2 3 4 5 6 3 -3 -2 -1
1 2 3 4 5 6 7 5 -2 -1
1 2 3 4 5 6 7 8 7 -1
1 2 3 4 5 6 7 8 9 9

=== Multiplicacion ===

=== Traspuesta ===
1 1 1 1 1 1 1 1 1 1
0 2 2 2 2 2 2 2 2 2
0 0 3 3 3 3 3 3 3 3
0 0 0 4 4 4 4 4 4 4
0 0 0 0 5 5 5 5 5 5
0 0 0 0 0 6 6 6 6 6
0 0 0 0 0 0 7 7 7 7
0 0 0 0 0 0 0 8 8 8
0 0 0 0 0 0 0 0 9 9
0 0 0 0 0 0 0 0 0 10

10 0 0 0 0 0 0 0 0 0
9 9 0 0 0 0 0 0 0 0
8 8 8 0 0 0 0 0 0 0
7 7 7 7 0 0 0 0 0 0
6 6 6 6 6 0 0 0 0 0

```

Podemos decir que los programas fueron ejecutados con distintos procesos tanto el de Linux como el de Windows ya que se tuvieron que hacer más líneas de código para poder compilarlo en Windows

Conclusiones

Al inicio se nos complicó un poco entender el funcionamiento de los procesos y cómo se pueden manejar, pero esta práctica sirvió bastante para resolver dudas en la parte teórica y aprender a cómo hacer una mejor programación de los procesos. Por ejemplo, el hecho de que tenemos que hacer una correcta terminación de los procesos es importante, debemos delimitar las instrucciones que le corresponden a cada uno ya que eso nos puede causar bastantes problemas con respecto a los resultados esperados. Algunos de los posibles problemas que se pueden presentar son la creación de procesos hijos por medio de un ciclo los cuales ni siquiera teníamos pensado crear o que un proceso padre lleve a cabo las instrucciones de un proceso hijo cuando eso no le corresponde. Para practicar estos problemas no llegan a más, pero si se quiere implementar en un software que después será usado para más usuarios entonces causará muchos conflictos que pueden ser difíciles de resolver en un futuro.

También se aprendió el uso de los árboles de procesos que prácticamente es como un árbol genealógico, pero con procesos, y en cada círculo que compone cada generación se incluye el PID del proceso. Esto ayuda bastante si es que no comprendemos cómo se enlaza un proceso con su proceso hijo y este a la vez con su propio proceso hijo, tanto si lo queremos implementar en un código como si queremos solo entender el concepto.

Bibliografía

- Daniel P. Bovet, Marco Cesati, Ed. O'Reilly Understanding the Linux Kernel (3ª Edición) Maxwell Remy Card 2005

