



INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO



LABORATORIO DE SISTEMAS OPERATIVOS

PRÁCTICA 4: **“Administrador de procesos en Linux y** **Windows”**

INTEGRANTES:

- Aguirre Cruz Eder Jonathan
- Buendía Moreno Hugo Vidal
- Saules Cortes Jhonatan

MATERIA: Sistemas Operativos

PROFESOR: Cortes Galicia Jorge

GRUPO: 2CM4

FECHA DE ENTREGA: 19- 06- 2015

Competencias

El alumno aprende a familiarizarse con el administrador de procesos del sistema operativo Linux, a través de la creación de nuevos procesos, ejecución y terminación

Desarrollo

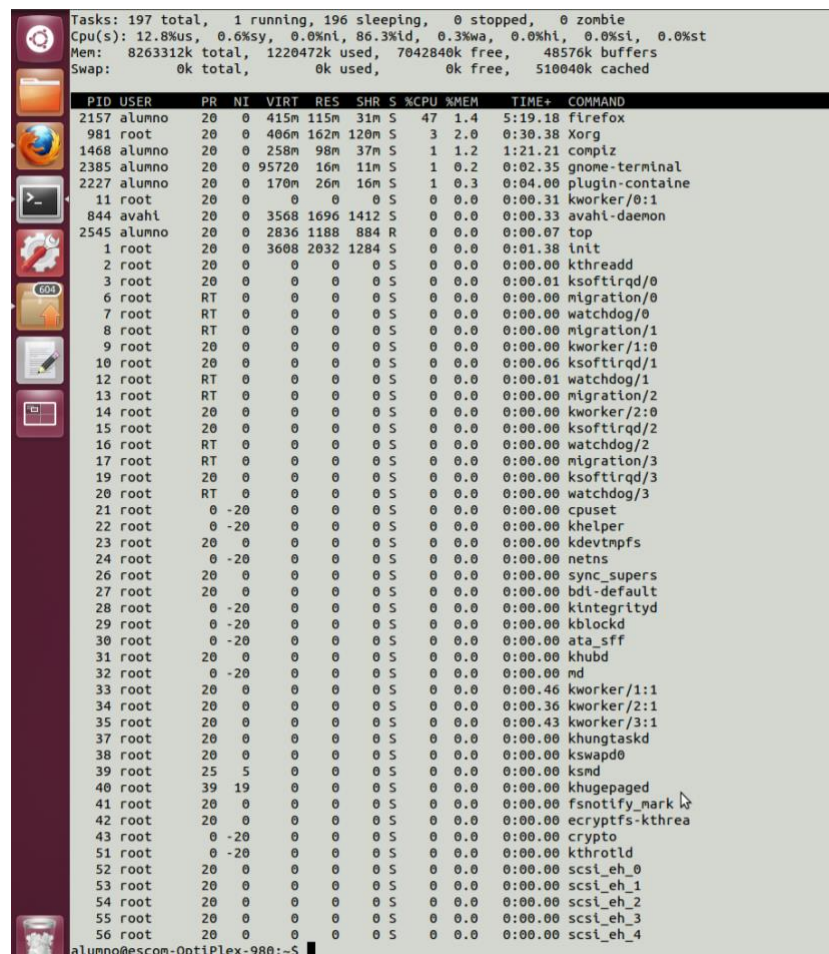
➤ Observaciones Individuales

Sección Linux

1.- Buendía Vidal Hugo

1. Introduzca los siguientes comandos a través del Shell del sistema operativo LINUX

Ps



```
Tasks: 197 total, 1 running, 196 sleeping, 0 stopped, 0 zombie
Cpu(s): 12.8%us, 0.6%sy, 0.0%ni, 86.3%id, 0.3%wa, 0.0%hi, 0.0%st, 0.0%st
Mem: 8263312k total, 1220472k used, 7042840k free, 48576k buffers
Swap: 0k total, 0k used, 0k free, 510040k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2157	alunno	20	0	415m	115m	31m	S	47	1.4	5:19.18	firefox
981	root	20	0	406m	162m	120m	S	3	2.0	0:30.38	Xorg
1468	alunno	20	0	258m	98m	37m	S	1	1.2	1:21.21	compiz
2385	alunno	20	0	95720	16m	11m	S	1	0.2	0:02.35	gnome-terminal
2227	alunno	20	0	170m	26m	16m	S	1	0.3	0:04.00	plugin-containe
11	root	20	0	0	0	0	S	0	0.0	0:00.31	kworker/0:1
844	avahi	20	0	3568	1696	1412	S	0	0.0	0:00.33	avahi-daemon
2545	alunno	20	0	2836	1188	884	R	0	0.0	0:00.07	top
1	root	20	0	3608	2032	1284	S	0	0.0	0:01.38	init
2	root	20	0	0	0	0	S	0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0	0.0	0:00.01	ksoftirqd/0
6	root	RT	0	0	0	0	S	0	0.0	0:00.00	migration/0
7	root	RT	0	0	0	0	S	0	0.0	0:00.00	watchdog/0
8	root	RT	0	0	0	0	S	0	0.0	0:00.00	migration/1
9	root	20	0	0	0	0	S	0	0.0	0:00.00	kworker/1:0
10	root	20	0	0	0	0	S	0	0.0	0:00.06	ksoftirqd/1
12	root	RT	0	0	0	0	S	0	0.0	0:00.01	watchdog/1
13	root	RT	0	0	0	0	S	0	0.0	0:00.00	migration/2
14	root	20	0	0	0	0	S	0	0.0	0:00.00	kworker/2:0
15	root	20	0	0	0	0	S	0	0.0	0:00.00	ksoftirqd/2
16	root	RT	0	0	0	0	S	0	0.0	0:00.00	watchdog/2
17	root	RT	0	0	0	0	S	0	0.0	0:00.00	migration/3
19	root	20	0	0	0	0	S	0	0.0	0:00.00	ksoftirqd/3
20	root	RT	0	0	0	0	S	0	0.0	0:00.00	watchdog/3
21	root	0	-20	0	0	0	S	0	0.0	0:00.00	cpuset
22	root	0	-20	0	0	0	S	0	0.0	0:00.00	khelper
23	root	20	0	0	0	0	S	0	0.0	0:00.00	kdevtmpfs
24	root	0	-20	0	0	0	S	0	0.0	0:00.00	netns
26	root	20	0	0	0	0	S	0	0.0	0:00.00	sync_supers
27	root	20	0	0	0	0	S	0	0.0	0:00.00	bdl-default
28	root	0	-20	0	0	0	S	0	0.0	0:00.00	kintegrityd
29	root	0	-20	0	0	0	S	0	0.0	0:00.00	kblockd
30	root	0	-20	0	0	0	S	0	0.0	0:00.00	ata_sff
31	root	20	0	0	0	0	S	0	0.0	0:00.00	khudb
32	root	0	-20	0	0	0	S	0	0.0	0:00.00	md
33	root	20	0	0	0	0	S	0	0.0	0:00.46	kworker/1:1
34	root	20	0	0	0	0	S	0	0.0	0:00.36	kworker/2:1
35	root	20	0	0	0	0	S	0	0.0	0:00.43	kworker/3:1
37	root	20	0	0	0	0	S	0	0.0	0:00.00	khungtaskd
38	root	20	0	0	0	0	S	0	0.0	0:00.00	kswapd0
39	root	25	5	0	0	0	S	0	0.0	0:00.00	ksmd
40	root	39	19	0	0	0	S	0	0.0	0:00.00	khugepaged
41	root	20	0	0	0	0	S	0	0.0	0:00.00	fnotify_nark
42	root	20	0	0	0	0	S	0	0.0	0:00.00	ecryptfs-kthrea
43	root	0	-20	0	0	0	S	0	0.0	0:00.00	crypto
51	root	0	-20	0	0	0	S	0	0.0	0:00.00	kthrotld
52	root	20	0	0	0	0	S	0	0.0	0:00.00	scsi_ah_0
53	root	20	0	0	0	0	S	0	0.0	0:00.00	scsi_ah_1
54	root	20	0	0	0	0	S	0	0.0	0:00.00	scsi_ah_2
55	root	20	0	0	0	0	S	0	0.0	0:00.00	scsi_ah_3
56	root	20	0	0	0	0	S	0	0.0	0:00.00	scsi_ah_4

alunno@escom-OptiPlex-980:~\$

Ps -fea

```
alumno@escom-OptiPlex-980: ~
2488 pts/0    00:00:00 ps
alumno@escom-OptiPlex-980:~$ ps -fea
  UID      PID  PPID  C  STIME TTY          TIME CMD
root         1      0  0  07:14 ?        00:00:01 /sbin/init
root         2      0  0  07:14 ?        00:00:00 [kthreadd]
root         3      2  0  07:14 ?        00:00:00 [ksoftirqd/0]
root         6      2  0  07:14 ?        00:00:00 [migration/0]
root         7      2  0  07:14 ?        00:00:00 [watchdog/0]
root         8      2  0  07:14 ?        00:00:00 [migration/1]
root         9      2  0  07:14 ?        00:00:00 [kworker/1:0]
root        10      2  0  07:14 ?        00:00:00 [ksoftirqd/1]
root        11      2  0  07:14 ?        00:00:00 [kworker/0:1]
root        12      2  0  07:14 ?        00:00:00 [watchdog/1]
root        13      2  0  07:14 ?        00:00:00 [migration/2]
root        14      2  0  07:14 ?        00:00:00 [kworker/2:0]
root        15      2  0  07:14 ?        00:00:00 [ksoftirqd/2]
root        16      2  0  07:14 ?        00:00:00 [watchdog/2]
root        17      2  0  07:14 ?        00:00:00 [migration/3]
root        19      2  0  07:14 ?        00:00:00 [ksoftirqd/3]
root        20      2  0  07:14 ?        00:00:00 [watchdog/3]
root        21      2  0  07:14 ?        00:00:00 [cpuset]
root        22      2  0  07:14 ?        00:00:00 [khelper]
root        23      2  0  07:14 ?        00:00:00 [kdevtmpfs]
root        24      2  0  07:14 ?        00:00:00 [netns]
root        26      2  0  07:14 ?        00:00:00 [sync_supers]
root        27      2  0  07:14 ?        00:00:00 [bdi-default]
root        28      2  0  07:14 ?        00:00:00 [kintegrityd]
root        29      2  0  07:14 ?        00:00:00 [kblockd]
root        30      2  0  07:14 ?        00:00:00 [ata_sff]
root        31      2  0  07:14 ?        00:00:00 [khubd]
root        32      2  0  07:14 ?        00:00:00 [nd]
root        33      2  0  07:14 ?        00:00:00 [kworker/1:1]
root        34      2  0  07:14 ?        00:00:00 [kworker/2:1]
root        35      2  0  07:14 ?        00:00:00 [kworker/3:1]
root        37      2  0  07:14 ?        00:00:00 [khungtaskd]
root        38      2  0  07:14 ?        00:00:00 [kswapd0]
root        39      2  0  07:14 ?        00:00:00 [ksmd]
root        40      2  0  07:14 ?        00:00:00 [khugepaged]
root        41      2  0  07:14 ?        00:00:00 [fsnotify_mark]
root        42      2  0  07:14 ?        00:00:00 [ecryptfs-kthrea]
root        43      2  0  07:14 ?        00:00:00 [crypto]
root        51      2  0  07:14 ?        00:00:00 [kthrotld]
root        52      2  0  07:14 ?        00:00:00 [scsi_eh_0]
root        53      2  0  07:14 ?        00:00:00 [scsi_eh_1]
root        54      2  0  07:14 ?        00:00:00 [scsi_eh_2]
root        55      2  0  07:14 ?        00:00:00 [scsi_eh_3]
root        56      2  0  07:14 ?        00:00:00 [scsi_eh_4]
root        57      2  0  07:14 ?        00:00:00 [scsi_eh_5]
root        61      2  0  07:14 ?        00:00:00 [kworker/u:5]
root        63      2  0  07:14 ?        00:00:00 [kworker/u:7]
root        82      2  0  07:14 ?        00:00:00 [devfreq_wq]
root        83      2  0  07:14 ?        00:00:00 [kworker/3:2]
root       254      2  0  07:14 ?        00:00:00 [jbd2/sda5-8]
root       255      2  0  07:14 ?        00:00:00 [ext4-dio-unwrit]
root       275      1  0  07:14 ?        00:00:00 mountall --daemon
root       339      1  0  07:14 ?        00:00:00 upstart-udev-bridge --daemon
root       368      1  0  07:14 ?        00:00:00 /sbin/udevd --daemon
root       475     368  0  07:14 ?        00:00:00 /sbin/udevd --daemon
```

¿Qué información le proporcionan los comandos anteriores?

Nos muestran la información de los procesos en ejecución del sistema operativo cuando se da el comando a la consola, se muestran datos como el id, el id del proceso padre. Con el comando `ps -fea` se muestra un poco más de información que con `ps`.

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 funciones **getpid()**, **getppid()** y **wait()** en la ayuda en línea, reporte sus observaciones.

SYNOPSIS

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

pid_t getpid(void);
pid_t getppid(void);
```

DESCRIPTION

getpid() returns the process ID of the calling process. (This is often used by routines that generate unique temporary filenames.)

getppid() returns the process ID of the parent of the calling process.

ERRORS

These functions are always successful.

CONFORMING TO

POSIX.1-2001, 4.3BSD, SVr4.

NOTES

Since glibc version 2.3.4, the glibc wrapper function for **getpid()** caches PIDs, so as to avoid additional system calls when a process calls **getpid()** repeatedly. Normally this caching is invisible, but its

Manual page getpid(2) line 8/50 48% (press h for help or q to quit)

WAIT(2)

Linux Programmer's Manual

WAIT(2)

NAME

wait, waitpid, waitid - wait for process to change state

SYNOPSIS

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

pid_t wait(int *status);

pid_t waitpid(pid_t pid, int *status, int options);

int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
```

Feature Test Macro Requirements for glibc (see **feature_test_macros(7)**):

```
waitid():
    _SVID_SOURCE || _XOPEN_SOURCE >= 500 ||
    _XOPEN_SOURCE && _XOPEN_SOURCE_EXTENDED
    || /* Since glibc 2.12: */ _POSIX_C_SOURCE >= 200809L
```

DESCRIPTION

Manual page wait(2) line 1 (press h for help or q to quit)

3.- Saules Cortés Jhonatan

3. Capture, compile y ejecute los 2 programas de creación de un nuevo proceso que a continuación se muestra. Observe su funcionamiento y experimente con el código

```
Terminal
jhonatan@jhonatan-HP-14-Notebook-PC: ~/Escritorio/Practica4
jhonatan@jhonatan-HP-14-Notebook-PC:~/Escritorio/Practica4$ gcc -o ejemplo1 Ejemplo1.c
jhonatan@jhonatan-HP-14-Notebook-PC:~/Escritorio/Practica4$ ./ejemplo1
Soy el proceso padre
Soy el proceso hijo
jhonatan@jhonatan-HP-14-Notebook-PC:~/Escritorio/Practica4$
```

```
Ejemplo1.c x
#include <stdlib.h>
#include <stdio.h>
#include <unistd.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);
    }
}
```

```
Terminal
jhonatan@jhonatan-HP-14-Notebook-PC: ~/Escritorio/Practica4
jhonatan@jhonatan-HP-14-Notebook-PC:~/Escritorio/Practica4$ gcc -o ejemplo2 Ejemplo2.c
jhonatan@jhonatan-HP-14-Notebook-PC:~/Escritorio/Practica4$ ./ejemplo2
Soy el proceso padre
Mensaje en ambos
Soy el proceso hijo
Mensaje en ambos
jhonatan@jhonatan-HP-14-Notebook-PC:~/Escritorio/Practica4$
```

```
Ejemplo1.c x Ejemplo2.c x
#include <stdlib.h>
#include <stdio.h>
#include <unistd.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);
}
```

4. Programe una aplicación que cree el árbol de procesos mostrado en el pizarrón. Para cada uno de los procesos creados se imprimirá en pantalla el pid de su padre si se trata de un hijo terminal o los pid's de sus hijos creados si se trata de un proceso padre. Dibuje en papel el árbol creado usando los pid's.

- Código Fuente

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(void){
    pid_t proc;
    int a,b,c,d,e,f,g;
    for(a=1;a<=10;a++){
        proc=fork();
        if(proc){
            printf("Soy el proceso
%d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
            wait(0);
            if(a==1){
                for(b=1;b<=8;b++){
                    proc=fork();
                    if(proc){
                        printf("Soy el proceso
%d\n",getpid());
                        wait(0);
                    }
                    else{
                        printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
                        wait(0);
                        if(b==1){
                            for(c=1;c<=6;c++){
                                proc=fork();
                                if(proc){
                                    printf("Soy el proceso
%d\n",getpid());
                                    wait(0);
                                }
                                else{
                                    printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
                                    wait(0);
                                    if(c==1){
                                        for(d=1;d<=4;d++){
                                            proc=fork();
                                            if(proc){
                                                printf("Soy el
proceso %d\n",getpid());
```

```
wait(0);
                                            }
                                            else{
                                                printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
                                                wait(0);
                                                if(d==1){
                                                    for(e=1;e<=2;e++){
                                                        proc=fork();
                                                        if(proc){
                                                            printf("Soy el
proceso %d\n",getpid());
                                                            wait(0);
                                                        }
                                                        else{
                                                            printf("soy el
hijo %d, mi padre es
%d\n",getpid(),getppid());
                                                            wait(0);
                                                            exit(0);
                                                        }
                                                        if(d==2){
                                                            proc=fork();
                                                            if(proc){
                                                                printf("Soy el
proceso %d\n",getpid());
                                                                wait(0);
                                                            }
                                                            else{
                                                                printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
                                                                wait(0);
                                                                exit(0);
                                                            }
                                                            if(d==3){
                                                                proc=fork();
                                                                if(proc){
                                                                    printf("Soy el
proceso %d\n",getpid());
                                                                    wait(0);
                                                                }
                                                                else{
                                                                    printf("soy el hijo
```

```
%d, mi padre es
%d\n",getpid(),getppid());
                                                                    wait(0);
                                                                    exit(0);
                                                                }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
            if(d==4){
                for(e=1;e<=2;e++){
                    proc=fork();
                    if(proc){
                        printf("Soy el
proceso %d\n",getpid());
                        wait(0);
                    }
                    else{
                        printf("soy el
hijo %d, mi padre es
%d\n",getpid(),getppid());
                        wait(0);
                        exit(0);
                    }
                    exit(0);
                }
                exit(0);
            }
            if(c==2){
                proc=fork();
                if(proc){
                    printf("Soy el
proceso %d\n",getpid());
                    wait(0);
                }
                else{
                    printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
                    wait(0);
                    proc=fork();
                    if(proc){
                        printf("Soy el
proceso %d\n",getpid());
                        wait(0);
                    }
                    else{
                        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
                        wait(0);
                        proc=fork();
                        if(proc){
                            printf("Soy el
proceso %d\n",getpid());
                            wait(0);
                        }
                        else{
                            printf("soy el hijo
```



```

%d, mi padre es
%d\n",getpid(),getppid());
    wait(0);
    exit(0);
}
}
exit(0);
}
}
if(c==3){
    proc=fork();
    if(proc){
        printf("Soy el
proceso %d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el
proceso %d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
            wait(0);
            exit(0);
        }
    }
}
if(c==4){
    proc=fork();
    if(proc){
        printf("Soy el
proceso %d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el
proceso %d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo

```

```

        }
        exit(0);
    }
}
if(c==5){
    proc=fork();
    if(proc){
        printf("Soy el
proceso %d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el
proceso %d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
            wait(0);
            exit(0);
        }
    }
}
if(c==6){
    for(d=1;d<=4;d++){
        proc=fork();
        if(proc){
            printf("Soy el
proceso %d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
            wait(0);
            if(d==1){
                for(e=1;e<=2;e++){
                    proc=fork();
                    if(proc){
                        printf("Soy el
proceso %d\n",getpid());
                        wait(0);
                    }
                    else{
                        printf("soy el
hijo %d, mi padre es
%d\n",getpid(),getppid());
                        wait(0);
                        exit(0);
                    }
                }
            }
        }
    }
}

```

```

        }
        exit(0);
    }
}
if(d==2){
    proc=fork();
    if(proc){
        printf("Soy el
proceso %d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        exit(0);
    }
}
if(d==3){
    proc=fork();
    if(proc){
        printf("Soy el
proceso %d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        exit(0);
    }
}
if(d==4){
    for(e=1;e<=2;e++){
        proc=fork();
        if(proc){
            printf("Soy el
proceso %d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el
hijo %d, mi padre es
%d\n",getpid(),getppid());
            wait(0);
            exit(0);
        }
    }
}
exit(0);
}
}
exit(0);
}
}

```

```

    }
    exit(0);
}
if(b==2){
    proc=fork();
    if(proc){
        printf("Soy el proceso
%d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el proceso
%d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
            wait(0);
            proc=fork();
            if(proc){
                printf("Soy el proceso
%d\n",getpid());
                wait(0);
            }
            else{
                printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
                wait(0);
                exit(0);
            }
        }
        exit(0);
    }
}
if(b==3){
    proc=fork();
    if(proc){
        printf("Soy el proceso
%d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el proceso
%d\n",getpid());

```

```

        wait(0);
    }
    else{
        printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el proceso
%d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
            wait(0);
            exit(0);
        }
    }
    exit(0);
}
if(b==4){
    proc=fork();
    if(proc){
        printf("Soy el proceso
%d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el proceso
%d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
            wait(0);
            proc=fork();
            if(proc){
                printf("Soy el proceso
%d\n",getpid());
                wait(0);
            }
            else{
                printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
                wait(0);
                exit(0);
            }
        }
    }
}

```

```

    }
    exit(0);
}
    exit(0);
}
if(b==5){
    proc=fork();
    if(proc){
        printf("Soy el proceso
%d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el proceso
%d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
            wait(0);
            proc=fork();
            if(proc){
                printf("Soy el proceso
%d\n",getpid());
                wait(0);
            }
            else{
                printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
                wait(0);
                exit(0);
            }
        }
    }
    exit(0);
}
if(b==1){
    for(c=1;c<=6;c++){
        proc=fork();
        if(proc){
            printf("Soy el proceso
%d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
            wait(0);

```



```

if(c==1){
    for(d=1;d<=4;d++){
        proc=fork();
        if(proc){
            printf("Soy el
proceso %d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
            wait(0);
            if(d==1){
                for(e=1;e<=2;e++){
                    proc=fork();
                    if(proc){
                        printf("Soy el
proceso %d\n",getpid());
                        wait(0);
                    }
                    else{
                        printf("soy el
hijo %d, mi padre es
%d\n",getpid(),getppid());
                        wait(0);
                        exit(0);
                    }
                }
                if(d==2){
                    proc=fork();
                    if(proc){
                        printf("Soy el
proceso %d\n",getpid());
                        wait(0);
                    }
                    else{
                        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
                        wait(0);
                        exit(0);
                    }
                }
                if(d==3){
                    proc=fork();
                    if(proc){
                        printf("Soy el
proceso %d\n",getpid());
                        wait(0);
                    }
                    else{
                        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
                        wait(0);
                        exit(0);
                    }
                }
            }
        }
    }
}

```

```

}
}
if(d==4){
    for(e=1;e<=2;e++){
        proc=fork();
        if(proc){
            printf("Soy el
proceso %d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el
hijo %d, mi padre es
%d\n",getpid(),getppid());
            wait(0);
            exit(0);
        }
    }
    exit(0);
}
exit(0);
}
}
if(c==2){
    proc=fork();
    if(proc){
        printf("Soy el
proceso %d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el
proceso %d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
            wait(0);
            exit(0);
        }
    }
    if(c==3){
        proc=fork();
        if(proc){
            printf("Soy el
proceso %d\n",getpid());
            wait(0);
        }
    }
}

```

```

}
else{
    printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
    wait(0);
    proc=fork();
    if(proc){
        printf("Soy el
proceso %d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        exit(0);
    }
}
if(c==4){
    proc=fork();
    if(proc){
        printf("Soy el
proceso %d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el
proceso %d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
            wait(0);
            exit(0);
        }
    }
}
if(c==5){
    proc=fork();
    if(proc){
        printf("Soy el
proceso %d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo
%d, mi padre es

```

```

%d\n",getpid(),getppid());
    wait(0);
    proc=fork();
    if(proc){
        printf("Soy el
proceso %d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        exit(0);
    }
}
if(c==6){
    for(d=1;d<=4;d++){
        proc=fork();
        if(proc){
            printf("Soy el
proceso %d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
            wait(0);
            if(d==1){
for(e=1;e<=2;e++){
    proc=fork();
    if(proc){
        printf("Soy el
proceso %d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el
hijo %d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        exit(0);
    }
}
exit(0);
}
if(d==2){
    proc=fork();
    if(proc){
        printf("Soy el
proceso %d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo
%d, mi padre es

```

```

%d\n",getpid(),getppid());
    wait(0);
    exit(0);
}
if(d==3){
    proc=fork();
    if(proc){
        printf("Soy el
proceso %d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        exit(0);
    }
}
if(d==4){
for(e=1;e<=2;e++){
    proc=fork();
    if(proc){
        printf("Soy el
proceso %d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el
hijo %d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        exit(0);
    }
}
exit(0);
}
exit(0);
}
exit(0);
}
exit(0);
}
exit(0);
}
exit(0);
}
if(a==2){
    proc=fork();
    if(proc){
        printf("Soy el proceso
%d\n",getpid());
        wait(0);
    }
}

```

```

}
else{
    printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
    wait(0);
    proc=fork();
    if(proc){
        printf("Soy el proceso
%d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el proceso
%d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
            wait(0);
            exit(0);
        }
        exit(0);
    }
}
exit(0);
}
if(a==3){
    proc=fork();
    if(proc){
        printf("Soy el proceso
%d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
        wait(0);
    }
}

```



```

printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
wait(0);
proc=fork();
if(proc){
printf("Soy el proceso
%d\n",getpid());
wait(0);
}
else{
printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
wait(0);
exit(0);
}
exit(0);
}
exit(0);
}
}
if(a==7){
proc=fork();
if(proc){
printf("Soy el proceso
%d\n",getpid());
wait(0);
}
else{
printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
wait(0);
proc=fork();
if(proc){
printf("Soy el proceso
%d\n",getpid());
wait(0);
}
else{
printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
wait(0);
proc=fork();
if(proc){
printf("Soy el proceso
%d\n",getpid());
wait(0);
}
else{
printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
wait(0);
proc=fork();
if(proc){

```

```

printf("Soy el proceso
%d\n",getpid());
wait(0);
}
else{
printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
wait(0);
exit(0);
}
exit(0);
}
}
if(a==8){
proc=fork();
if(proc){
printf("Soy el proceso
%d\n",getpid());
wait(0);
}
else{
printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
wait(0);
proc=fork();
if(proc){
printf("Soy el proceso
%d\n",getpid());
wait(0);
}
else{
printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
wait(0);
proc=fork();
if(proc){
printf("Soy el proceso
%d\n",getpid());
wait(0);
}
else{
printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
wait(0);
proc=fork();
if(proc){

```

```

padre es
%d\n",getpid(),getppid());
wait(0);
exit(0);
}
exit(0);
}
}
if(a==9){
proc=fork();
if(proc){
printf("Soy el proceso
%d\n",getpid());
wait(0);
}
else{
printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
wait(0);
proc=fork();
if(proc){
printf("Soy el proceso
%d\n",getpid());
wait(0);
}
else{
printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
wait(0);
proc=fork();
if(proc){
printf("Soy el proceso
%d\n",getpid());
wait(0);
}
else{
printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
wait(0);
proc=fork();
if(proc){

```

```

    }
    exit(0);
}
exit(0);
}
}
if(a==10){
    for(b=1;b<=8;b++){
        proc=fork();
        if(proc){
            printf("Soy el proceso
%d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
            wait(0);
            if(b==1){
                for(c=1;c<=6;c++){
                    proc=fork();
                    if(proc){
                        printf("Soy el proceso
%d\n",getpid());
                        wait(0);
                    }
                    else{
                        printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
                        wait(0);
                        if(c==1){
                            for(d=1;d<=4;d++){
                                proc=fork();
                                if(proc){
                                    printf("Soy el
proceso %d\n",getpid());
                                    wait(0);
                                }
                                else{
                                    printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
                                    wait(0);
                                    if(d==1){
                                        for(e=1;e<=2;e++){
                                            proc=fork();
                                            if(proc){
                                                printf("Soy el
proceso %d\n",getpid());
                                                wait(0);
                                            }
                                            else{
                                                printf("soy el
hijo %d, mi padre es
%d\n",getpid(),getppid());
                                                wait(0);
                                                exit(0);
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
    exit(0);
}
if(d==2){
    proc=fork();
    if(proc){
        printf("Soy el
proceso %d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        exit(0);
    }
}
if(d==3){
    proc=fork();
    if(proc){
        printf("Soy el
proceso %d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        exit(0);
    }
}
if(d==4){
    for(e=1;e<=2;e++){
        proc=fork();
        if(proc){
            printf("Soy el
proceso %d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el
hijo %d, mi padre es
%d\n",getpid(),getppid());
            wait(0);
            exit(0);
        }
    }
    exit(0);
}
}
exit(0);
}
}
if(c==2){
    proc=fork();
}

```

```

    if(proc){
        printf("Soy el
proceso %d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el
proceso %d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
            wait(0);
            exit(0);
        }
    }
    if(c==3){
        proc=fork();
        if(proc){
            printf("Soy el
proceso %d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
            wait(0);
            proc=fork();
            if(proc){
                printf("Soy el
proceso %d\n",getpid());
                wait(0);
            }
            else{
                printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
                wait(0);
                exit(0);
            }
        }
    }
    if(c==4){
        proc=fork();
        if(proc){
            printf("Soy el
proceso %d\n",getpid());
            wait(0);
        }
    }
}

```

```

    }
    else{
        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el
proceso %d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
            wait(0);
            exit(0);
        }
    }
    if(c==5){
        proc=fork();
        if(proc){
            printf("Soy el
proceso %d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
            wait(0);
            proc=fork();
            if(proc){
                printf("Soy el
proceso %d\n",getpid());
                wait(0);
            }
            else{
                printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
                wait(0);
                exit(0);
            }
        }
    }
    if(c==6){
        for(d=1;d<=4;d++){
            proc=fork();
            if(proc){
                printf("Soy el
proceso %d\n",getpid());
                wait(0);
            }
            else{
                printf("soy el hijo

```

```

%d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        if(d==1){
            for(e=1;e<=2;e++){
                proc=fork();
                if(proc){
                    printf("Soy el
proceso %d\n",getpid());
                    wait(0);
                }
                else{
                    printf("soy el
hijo %d, mi padre es
%d\n",getpid(),getppid());
                    wait(0);
                    exit(0);
                }
            }
            exit(0);
        }
        if(d==2){
            proc=fork();
            if(proc){
                printf("Soy el
proceso %d\n",getpid());
                wait(0);
            }
            else{
                printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
                wait(0);
                exit(0);
            }
        }
        if(d==3){
            proc=fork();
            if(proc){
                printf("Soy el
proceso %d\n",getpid());
                wait(0);
            }
            else{
                printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
                wait(0);
                exit(0);
            }
        }
        if(d==4){
            for(e=1;e<=2;e++){
                proc=fork();
                if(proc){
                    printf("Soy el
proceso %d\n",getpid());
                    wait(0);
                }
            }
        }
    }
}

```

```

    }
    else{
        printf("soy el
hijo %d, mi padre es
%d\n",getpid(),getppid());
        wait(0);
        exit(0);
    }
    exit(0);
}
exit(0);
}
exit(0);
}
exit(0);
}
if(b==2){
    proc=fork();
    if(proc){
        printf("Soy el proceso
%d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el proceso
%d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
            wait(0);
            proc=fork();
            if(proc){
                printf("Soy el proceso
%d\n",getpid());
                wait(0);
            }
            else{
                printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
                wait(0);
                exit(0);
            }
        }
    }
    exit(0);
}
}

```



```

    }
}
if(b==3){
    proc=fork();
    if(proc){
        printf("Soy el proceso
%d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el proceso
%d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
            wait(0);
            proc=fork();
            if(proc){
                printf("Soy el proceso
%d\n",getpid());
                wait(0);
            }
            else{
                printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
                wait(0);
                exit(0);
            }
        }
        exit(0);
    }
}
if(b==4){
    proc=fork();
    if(proc){
        printf("Soy el proceso
%d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el proceso

```

```

        }
    }
    else{
        printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el proceso
%d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
            wait(0);
            exit(0);
        }
    }
    exit(0);
}
exit(0);
}
if(b==5){
    proc=fork();
    if(proc){
        printf("Soy el proceso
%d\n",getpid());
        wait(0);
    }
    else{
        printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
        wait(0);
        proc=fork();
        if(proc){
            printf("Soy el proceso
%d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
            wait(0);
            proc=fork();
            if(proc){
                printf("Soy el proceso

```

```

        exit(0);
    }
    exit(0);
}
if(b==1){
    for(c=1;c<=6;c++){
        proc=fork();
        if(proc){
            printf("Soy el proceso
%d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo %d,
mi padre es
%d\n",getpid(),getppid());
            wait(0);
            if(c==1){
                for(d=1;d<=4;d++){
                    proc=fork();
                    if(proc){
                        printf("Soy el
proceso %d\n",getpid());
                        wait(0);
                    }
                    else{
                        printf("soy el hijo
%d, mi padre es
%d\n",getpid(),getppid());
                        wait(0);
                        if(d==1){
                            for(e=1;e<=2;e++){
                                proc=fork();
                                if(proc){
                                    printf("Soy el
proceso %d\n",getpid());
                                    wait(0);
                                }
                                else{
                                    printf("soy el
hijo %d, mi padre es
%d\n",getpid(),getppid());
                                    wait(0);
                                    exit(0);
                                }
                            }
                            exit(0);
                        }
                        if(d==2){
                            proc=fork();
                            if(proc){
                                printf("Soy el
proceso %d\n",getpid());
                                wait(0);
                            }
                            else{
                                printf("soy el hijo

```


inversas. Cada uno de estos procesos escribirá en un archivo diferente los resultados de la operación que realizó, con el sexto proceso se leerán los archivos de resultados y los mostrará en pantalla cada uno de ellos

- **Código fuente**

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

void SumaMatrices(int a[15],int
b[15],int r[15]);
void RestaMatrices(int a[15],int
b[15],int r[15]);
void MultiplicaMatrices(int
a[15],int b[15],int r[15]);
void TraspuestaMatriz(int a[15]);
int InversaMatriz(int a[15], float
matrizDelInversa[15]);
void ArchPutsMatriz(FILE
*Archivo, int Matriz[15]);
void ArchPutsMatrizFloat(FILE
*Archivo, float Matriz[15]);
void ArchImprimeMatriz(FILE
*Archivo);
void ArchImprimeContenido(FILE
*Archivo);
int main(int argc, char* argv){
    int proc,i,j;
    int matriz1[15][15],
matriz2[15][15];
    FILE
*ArchMatriz1=fopen("mat1.txt","r"
);
    FILE
*ArchMatriz2=fopen("mat2.txt","r"
);
    if (ArchMatriz1==NULL ||
ArchMatriz2==NULL){
        printf("Error al cargar los
archivos\n");
        exit(0);
    }
    //Obtener de los archivos las
matrices
    for(i=0; i<15; i++){
        for(j=0; j<15; j++){
            fscanf(ArchMatriz1, "%d",&matriz1
[i][j]);
            fscanf(ArchMatriz2, "%d",&matriz2
[i][j]);
```

```
        }
    }
    fclose(ArchMatriz1);
    fclose(ArchMatriz2);
    for(i=1; i<=6; i++){
        proc=fork();
        if(proc){
            printf("Soy el proceso
%d\n",getpid());
            wait(0);
        }
        else{
            printf("soy el hijo %d, mi
padre es
%d\n",getpid(),getppid());
            wait(0);
            //exit(0);
            if(i==1){
                //Suma de dos matrices de
15x15
                FILE
*ArchMatricesSumadas=fopen("
MatricesSumadas","w");
                int
matricesSumadas[15][15];
                SumaMatrices(matriz1,ma
triz2,matricesSumadas);
                ArchPutsMatriz(ArchMatri
cesSumadas,matricesSumadas);
                wait(0);
                exit(0);
            }
            if(i==2){
                //Resta de dos matrices de
15x15
                FILE
*ArchMatricesRestadas=fopen("
MatricesRestadas","w");
                int
matricesRestadas[15][15];
                RestaMatrices(matriz1,ma
triz2,matricesRestadas);
                ArchPutsMatriz(ArchMatri
cesRestadas,matricesRestadas);
                wait(0);
                exit(0);
            }
        }
    }
```

```

    if(i==3){
        //Multiplicacion de dos
        matrices de 15x15
        FILE
        *ArchMatricesMultiplicadas=fopen(
n("MatricesMultiplicadas","w");
        int matricesMulti[15][15];
        MultiplicaMatrices(matriz1
,matriz2,matricesMulti);
        ArchPutsMatriz(ArchMatri
cesMultiplicadas,matricesMulti);
        wait(0);
        exit(0);
    }
    if(i==4){
        //Transpuestas de las dos
        matrices
        FILE
        *ArchMatriz1Traspuesta=fopen("
Matriz1Traspuesta","w");
        TraspuestaMatriz(matriz1);
        FILE
        *ArchMatriz2Traspuesta=fopen("
Matriz2Traspuesta","w");
        TraspuestaMatriz(matriz2);
        ArchPutsMatriz(ArchMatriz
1Traspuesta,matriz1);
        ArchPutsMatriz(ArchMatriz
2Traspuesta,matriz2);
        wait(0);
        exit(0);
    }
    if(i==5){
        //Matrices inversas
        float matrizinv1[15][15],
matrizinv2[15][15];
        FILE
        *ArchMatriz1Inversa=fopen("Matri
z1Inversa","w");
        FILE
        *ArchMatriz2Inversa=fopen("Matri
z2Inversa","w");
        if(InversaMatriz(matriz1,m
atrizinv1)==1){

ArchPutsMatrizFloat(ArchMatriz1I
nversa,matrizinv1);
        }
        else{

fprintf(ArchMatriz1Inversa,"No
tiene inversa");
        }
        if(InversaMatriz(matriz2,m

```

```

atrizinv2)==1){

ArchPutsMatrizFloat(ArchMatriz2I
nversa,matrizinv2);
        }
        else{

fprintf(ArchMatriz2Inversa,"No
tiene inversa");
        }
        wait(0);
        exit(0);
    }
    if(i==6){
        //Lector y mostrar los
        resultados de los archivos
        creados

        //imprimir resultados
        printf("La suma de las
matrices es: \n");
        FILE
        *ArchMatricesSumadas=fopen("
MatricesSumadas","r");
        ArchImprimeMatriz(ArchM
atricesSumadas);
        printf("La resta de las
matrices es: \n");
        FILE
        *ArchMatricesRestadas=fopen("
MatricesRestadas","r");
        ArchImprimeMatriz(ArchM
atricesRestadas);
        printf("La multiplicacion
de las matrices es: \n");
        FILE
        *ArchMatricesMultiplicadas=fopen(
n("MatricesMultiplicadas","r");
        ArchImprimeMatriz(ArchM
atricesMultiplicadas);
        printf("Sus respectivas
traspuestas son: \n");
        FILE
        *ArchMatriz1Traspuesta=fopen("
Matriz1Traspuesta","r");
        FILE
        *ArchMatriz2Traspuesta=fopen("
Matriz2Traspuesta","r");
        printf("Primera matriz:
\n");
        ArchImprimeMatriz(ArchM
atriz1Traspuesta);
        printf("Segunda matriz:
\n");

```

```

        ArchImprimeMatriz(ArchMatriz2Traspuesta);
        printf("Sus respectivas inversas son: \n");
        FILE
        *ArchMatriz1Inversa=fopen("Matriz1Inversa","r");
        FILE
        *ArchMatriz2Inversa=fopen("Matriz2Inversa","r");
        printf("Primera matriz: \n");
        ArchImprimeContenido(ArchMatriz1Inversa);
        printf("Segunda matriz: \n");
        ArchImprimeContenido(ArchMatriz2Inversa);
        wait(0);
        exit(0);
    }
}
}
}
}
void SumaMatrices(int a[15],int b[15],int r[15]){
    int i,j;
    for (i=0;i<15;i++){
        for(j=0;j<15;j++){
            r[i][j]=a[i][j] + b[i][j];
        }
    }
}
void RestaMatrices(int a[15],int b[15],int r[15]){
    int i,j;
    for (i=0;i<15;i++){
        for(j=0;j<15;j++){
            r[i][j]=a[i][j] - b[i][j];
        }
    }
}
void MultiplicaMatrices(int a[15],int b[15],int r[15]){
    int i,j,k;
    for(i=0;i<15;i++){
        for (j=0;j<15;j++){
            r[i][j]=0;
            for(k=0;k<15;k++){
                r[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}

```

```

    }
}

void TraspuestaMatriz(int a[15][15]){
    int i,j,a1[15][15],b1[15][15];
    for (i=0;i<15;i++){
        for(j=0;j<15;j++){
            a1[i][j]=a[j][i];
        }
    }
    for (i=0;i<15;i++){
        for(j=0;j<15;j++){
            a[i][j]=a1[j][i];
        }
    }
}

int InversaMatriz(int a[15], float matrizDelInversa[15][15]){
    int i, j, k, terminosIguales=0, orden=15;
    float pivote=0,
    matrizDelIdentidad[orden][orden],
    ,dividendo=0;
    //Iniciando matriz identidad

    for(i=0;i<orden;i++){
        for(j=0;j<orden;j++){
            if(i==j){
                matrizDelInversa[i][j]=1;
            }
            else{
                matrizDelInversa[i][j]=0;
            }
        }
    }
    //Copiar matriz recibida entera en la flotante
    for(i=0;i<orden;i++){
        for(j=0;j<orden;j++){
            matrizDelIdentidad[i][j]=(float)a[i][j];
        }
    }
    //Iniciar el algoritmo de obtener la inversa por Gauss Jordan
    //Volviendola triangular inferior
    for(i=0;i<orden;i++){
        pivote=matrizDelIdentidad[i][i];
        for(j=i;j<orden-1;j++){
            dividendo=
            matrizDelIdentidad[j+1][i];
            terminosIguales=0;

```



```

    for(k=0;k<orden;k++){
        if(matrizDeldentidad[j+1][k]
]==matrizDeldentidad[i][k]){
            terminosIguales++;

        if(terminosIguales==orden){
            return 0;
        }
    }
    matrizDeldentidad[j+1][k]=
((pivote/dividendo)*
matrizDeldentidad[j+1][k])-
matrizDeldentidad[i][k];
    matrizDelInversa[j+1][k]=(
(pivote/dividendo)*
matrizDelInversa[j+1][k])-
matrizDelInversa[i][k];
    }
}
}
//Volviendola triangular superior
y asi volverla una matriz escalar
(o puede que ya de identidad)
for(i=orden-1;0<=i;i--){
    pivote=matrizDeldentidad[i][i];
    for(j=i;0<=j;j--){
        dividendo=
matrizDeldentidad[j-1][i];
        for(k=orden-1;0<=k;k--){
            matrizDeldentidad[j-
1][k]=( (dividendo/pivote)*
matrizDeldentidad[i][k])-
matrizDeldentidad[j-1][k];
            matrizDelInversa[j-1][k]=(
(dividendo/pivote)*
matrizDelInversa[i][k])-
matrizDelInversa[j-1][k];
        }
    }
}
//Finalmente, la matriz inversa la
volvemos identidad dividiendola
si es escalar
for(i=0;i<orden;i++){
    for(j=0;j<orden;j++){

matrizDelInversa[i][j]=matrizDelInv
ersa[i][j]/matrizDeldentidad[i][i];
    }
}
}

```

```

return 1;
}

void ArchPutsMatriz(FILE
*Archivo, int Matriz[15][15]){
    int i,j;
    for (i=0;i<15;i++){
        for(j=0;j<15;j++){
            fprintf(Archivo,"%d ",Matriz
[i][j]);
        }
        fprintf(Archivo,"\n");
    }
}

void ArchPutsMatrizFloat(FILE
*Archivo, float Matriz[15][15]){
    int i,j;
    for (i=0;i<15;i++){
        for(j=0;j<15;j++){
            fprintf(Archivo,"%f ",Matriz
[i][j]);
        }
        fprintf(Archivo,"\n");
    }
}

void ArchImprimeMatriz(FILE
*Archivo){
    int i,j,a[15][15];
    for(i=0; i<15; i++){
        for(j=0; j<15; j++){
            fscanf(Archivo,"%d",&a[i][j]);
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
}

void ArchImprimeContenido(FILE
*Archivo){
    char Caracter;
    while(!feof(Archivo)){

fscanf(Archivo,"%c",&Caracter);
    printf("%c",Caracter);
    }
    printf("\n");
}
}

```

- **Capturas de pantalla**

[illegible]

```
Terminal
jhonatan@jhonatan-HP-14-Notebook-PC: ~/Escritorio/Practica4

La resta de las matrices es:
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 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 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 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 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 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 0 0 0 0 0 0

La multiplicacion de las matrices es:
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800
120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800

Sus respectivas traspuestas son:
Primera matriz:
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
```

```

Editor de textos
● ● ● jhonatan@jhonatan-HP-14-Notebook-PC: ~/Escritorio/Practica4

Sus respectivas traspuestas son:
Primera matriz:
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15

Segunda matriz:
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15

Sus respectivas inversas son:
Primera matriz:
No tiene Inversa
Segunda matriz:
No tiene Inversa
jhonatan@jhonatan-HP-14-Notebook-PC: ~/Escritorio/Practica4$

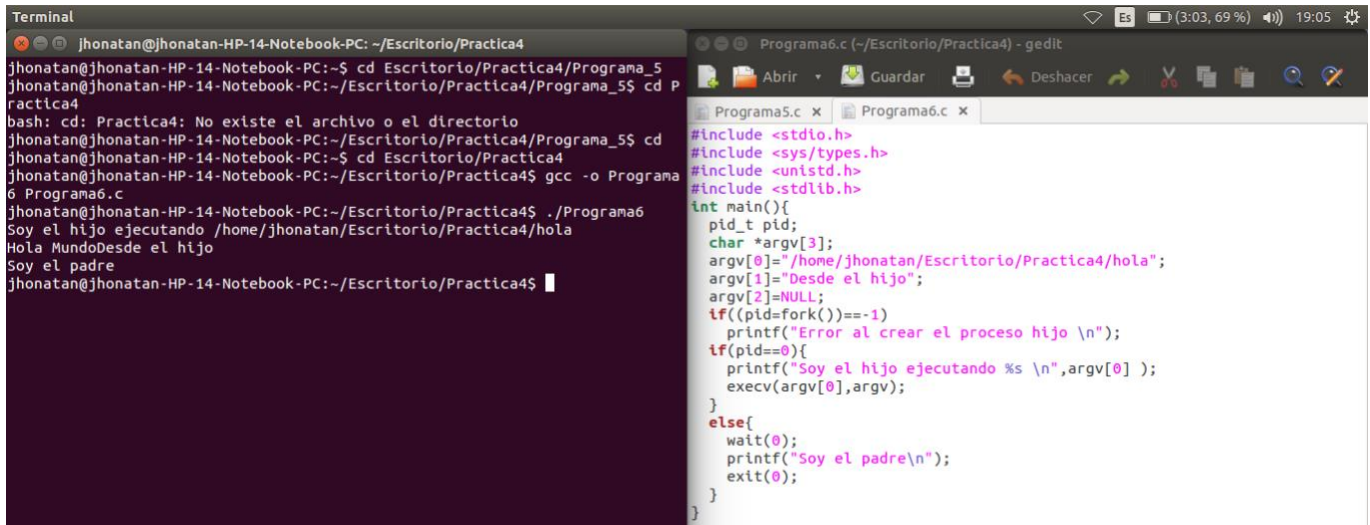
```

6.- Capture,
siguiente
de un nuevo

compile y ejecute el
programa de creación
proceso con sustitución

de código, así como el programa que será el nuevo código a ejecutar. Observe su funcionamiento y experimente con el código.

- **Código fuente**



The screenshot shows a terminal window on the left and a code editor on the right. The terminal displays the execution of a program named 'Programa6.c'. The user navigates to the directory 'Practica4/Programa_5' and runs 'gcc -o Programa6 Programa6.c'. Then, they run './Programa6', which outputs 'Soy el hijo ejecutando /home/jhonatan/Escritorio/Practica4/hola', 'Hola MundoDesde el hijo', and 'Soy el padre'. The code editor shows the source code of 'Programa6.c', which includes headers for stdio, sys/types,unistd, and stdlib. The main function uses fork to create a child process, prints error messages, and then prints the execution path and parent status.

7.- Programe una aplicación que cree un proceso hijo a partir de un proceso padre, el hijo creado a su vez creará tres hijos más. Cada uno de los tres procesos generados ejecutará tres programas diferentes mediante sustitución de código, el primer programa evaluará una expresión aritmética (ésta aplicación es la que programó en la práctica 1), el segundo programa cambiara los permisos de un archivo dado (ésta aplicación es la que programo en la practica 2), y el tercer programa obtendrá las matrices inversas que programó anteriormente. Observe el funcionamiento de su programa detalladamente y responda la siguiente pregunta ¿Es posible un funcionamiento 100% concurrente? De su aplicación. Explique porque si o no es 100% concurrente su aplicación.

- **Código fuente**

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

```
int main(){
    pid_t pid1, pid2;
    int i, j, k;
    char *argv1[3];
    char *argv2[3];
    char *argv3[3];
```

```
argv1[0]="/home/jhonatan/Escritorio/Practica4/Programa7/arbol";
```

```
argv1[1]="Desde el hijo 1";
argv1[2]=NULL;
```

```
argv2[0]="/home/jhonatan/Escritorio/Practica4/Programa7/permisos";
```

```
argv2[1]="Desde el hijo 2";
argv2[2]=NULL;
```

```
argv3[0]="/home/jhonatan/Escritorio/Practica4/Programa7/matriz_inversa";
```

```
argv3[1]="Desde el hijo 3";
argv3[2]=NULL;
```

```

pid1=fork();
if(pid1==0){
    for (i=0;i<3;i++){
        pid2=fork();
        if(pid2==0 && i==0){
            printf("Soy el hijo ejecutando
%s \n",argv1[0] );
            execv(argv1[0],argv1);
            //exit(0);
        }
        else if (pid2 !=0 && i==0) {
            wait(0);
            printf("SOY EL PAPA DEL
PRIMER PROCESO \n");
            //exit(0);
        }
        else if(pid2==0 && i==1){
            wait(NULL);
            printf("Soy el hijo 2
ejecutando %s \n",argv2[0] );
            execv(argv2[0],argv2);
            //exit(0);
        }
        else if (pid2!=0 && i==1){
            wait(0);

```

```

        printf("SOY EL PADRE DEL
SEGUNDO PROCESO\n");
        //exit(0);
    }
    else if(pid2==0 && i==2){
        wait(NULL);
        printf("Soy el hijo 3
ejecutando %s \n",argv3[0] );
        execv(argv3[0],argv3);
        //exit(0);
    }
    else if (pid2!=0 && i==2){
        wait(0);
        printf("SOY EL PADRE DEL
TECER PROCESO\n");
        exit(0);
    }
}
}
}
else{
    wait(0);
    printf("SOY EL SUPER PADRE\n");
    exit(0);
}
}
}

```

Matriz inversa

```

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

int InversaMatriz(int a[][15], float
matrizDelInversa[][15]);
void ArchPutsMatriz(FILE *Archivo,
int Matriz[][15]);
void ArchPutsMatrizFloat(FILE
*Archivo, float Matriz[][15]);
void ArchImprimeMatriz(FILE
*Archivo);
void ArchImprimeContenido(FILE
*Archivo);
int main(int argc, char* argv[]){
    int proc,i,j;
    int matriz1[15][15],
matriz2[15][15];
    FILE
*ArchMatriz1=fopen("mat1.txt","r"
);
    FILE
*ArchMatriz2=fopen("mat2.txt","r"
);
    if (ArchMatriz1==NULL ||

```

```

ArchMatriz2==NULL){
    printf("Error al cargar los
archivos\n");
    exit(0);
}
//Obtener de los archivos las
matrices
for(i=0; i<15; i++){
    for(j=0; j<15; j++){
        fscanf(ArchMatriz1,"%d",&matriz1[
i][j]);
        fscanf(ArchMatriz2,"%d",&matriz2[
i][j]);
    }
}
fclose(ArchMatriz1);
fclose(ArchMatriz2);
//Matrices inversas
float matrizinv1[15][15],
matrizinv2[15][15];
FILE
*ArchMatriz1Inversa=fopen("Matri
z1Inversa","w");

```

```

FILE
*ArchMatriz2Inversa=fopen("Matriz2Inversa","w");

if(InversaMatriz(matriz1,matrizinv1)==1){

ArchPutsMatrizFloat(ArchMatriz1Inversa,matrizinv1);
}
else{
fprintf(ArchMatriz1Inversa,"No tiene inversa");
}

if(InversaMatriz(matriz2,matrizinv2)==1){

ArchPutsMatrizFloat(ArchMatriz2Inversa,matrizinv2);
}
else{
fprintf(ArchMatriz2Inversa,"No tiene inversa");
}

fclose(ArchMatriz1Inversa);
fclose(ArchMatriz2Inversa);
printf("Sus respectivas inversas son: \n");

ArchMatriz1Inversa=fopen("Matriz1Inversa","r");

ArchMatriz2Inversa=fopen("Matriz2Inversa","r");
printf("Primera matriz: \n");

ArchImprimeContenido(ArchMatriz1Inversa);
printf("Segunda matriz: \n");

ArchImprimeContenido(ArchMatriz2Inversa);
}

int InversaMatriz(int a[15], float matrizDelInversa[15])
{
int i, j, k, terminosIguales=0, orden=15;
float pivote=0,
matrizDeldentidad[orden][orden],
dividendo=0;
//Inicializando matriz identidad

```

```

for(i=0;i<orden;i++){
for(j=0;j<orden;j++){
if(i==j){
matrizDelInversa[i][j]=1;
}
else{
matrizDelInversa[i][j]=0;
}
}
}
//Copiar matriz recibida entera en la flotante
for(i=0;i<orden;i++){
for(j=0;j<orden;j++){

matrizDeldentidad[i][j]=(float)a[i][j];
}
}
//Iniciar el algoritmo de obtener la inversa por Gauss Jordan
//Volviendola triangular inferior
for(i=0;i<orden;i++){
pivote=matrizDeldentidad[i][i];
for(j=i;j<orden-1;j++){
dividendo=
matrizDeldentidad[j+1][i];
terminosIguales=0;
for(k=0;k<orden;k++){
if(matrizDeldentidad[j+1][k]
]==matrizDeldentidad[i][k]){
terminosIguales++;
}
}
matrizDeldentidad[j+1][k]=
((pivote/dividendo)*
matrizDeldentidad[j+1][k])-
matrizDeldentidad[i][k];
matrizDelInversa[j+1][k]=(
(pivote/dividendo)*
matrizDelInversa[j+1][k])-
matrizDelInversa[i][k];
}
}
}
//Volviendola triangular superior
y asi volverla una matriz escalar
(o puede que ya de identidad)
for(i=orden-1;0<=i;i--){
pivote=matrizDeldentidad[i][i];
for(j=i;0<j;j--){

```

```

        dividendo=
matrizDeldentidad[j-1][i];
        for(k=orden-1;0<=k;k--){
            matrizDeldentidad[j-
1][k]=( (dividendo/pivote) *
matrizDeldentidad[i][k] )-
matrizDeldentidad[j-1][k];
            matrizDelInversa[j-1][k]=(
(dividendo/pivote) *
matrizDelInversa[i][k] )-
matrizDelInversa[j-1][k];
        }
    }
}

//Finalmente, la matriz inversa la
volvemos identidad dividiendola
si es escalar
for(i=0;i<orden;i++){
    for(j=0;j<orden;j++){

matrizDelInversa[i][j]=matrizDelInve
rsa[i][j]/matrizDeldentidad[i][i];
    }
}
return 1;
}

void ArchPutsMatriz(FILE *Archivo,
int Matriz[15][15]){
    int i,j;
    for (i=0;i<15;i++){
        for(j=0;j<15;j++){
            fprintf(Archivo,"%d ",Matriz
[i][j]);
        }
        fprintf(Archivo,"\n");
    }
}

```

```

}

void ArchPutsMatrizFloat(FILE
*Archivo, float Matriz[15][15]){
    int i,j;
    for (i=0;i<15;i++){
        for(j=0;j<15;j++){
            fprintf(Archivo,"%f ",Matriz
[i][j]);
        }
        fprintf(Archivo,"\n");
    }
}

void ArchImprimeMatriz(FILE
*Archivo){
    int i,j,a[15][15];
    for(i=0; i<15; i++){
        for(j=0; j<15; j++){
            fscanf(Archivo,"%d",&a[i][j]);
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
}

void ArchImprimeContenido(FILE
*Archivo){
    char Caracter;
    while(!feof(Archivo)){

fscanf(Archivo,"%c",&Caracter);
    printf("%c",Caracter);
    }
    printf("\n");
}
}

```

Permisos

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <dirent.h>
#define ARCHIVOS 20
#define CADENA 100
#define RUTA 300

int main(){
    int i, j, n, resp, fichero;

```

```

    DIR *dir;
    char c;
    char *aux[RUTA];
    unsigned int modo_permisos;
    struct dirent* dirp;
    struct stat listado;
    char ruta[RUTA];
    char copiaArchivo[CADENA];
    char copiaArchivo2[CADENA];
    char
*name_f[ARCHIVOS]={ "Archivo1.t
xt", "Archivo2.txt",
"Archivo3.txt", "Archivo4.txt", "Arch
ivo5.txt", "Archivo6.txt",

```



```
"Archivo7.txt","Archivo8.txt","Arch  
ivo9.txt","Archivo10.txt",
```

```
"Archivo11.txt","Archivo12.txt","Ar  
chivo13.txt","Archivo14.txt",
```

```
"Archivo15.txt","Archivo16.txt","Ar  
chivo17.txt","Archivo18.txt",  
"Archivo19.txt","Archivo20.txt");
```

```
char *conte[ARCHIVOS]={  
"Ciclistas en sentido contrario",  
"Bloqueo de la autopista  
Mexico Cuernavaca",  
"En el mar la vida es mas  
sabrosa",  
"Reductores de velocidad",  
"Falta de educacion vial",  
"Estacionarse en doble fila no  
es correcto",  
"Tope mal ubicado",  
"A que no puedes comer solo  
una",  
"Borracho manejo mejor",  
"En la sierra y en la ciudad",  
"HTML significaÂ HyperText  
Markup Language",  
"SGML (Standard Generalized  
Markup Language)",  
"Un sistema operativo  
desempeÃ±a 5 funciones  
bÃ¡sicas",  
"Las aplicaciones invocan  
estas rutinas a travÃ©s del uso de  
llamadas al sistema  
especÃ­ficas",  
"base: Define la ruta de  
acceso",  
"link: Define archivos  
vinculados",  
"meta: Define metadatos",
```

```
"script: Delimita el script  
incluido",
```

```
"style: Delimita definicion del  
estilo",
```

```
"Fin"
```

```
};
```

```
srand(time(NULL));
```

```
n = rand() % ARCHIVOS + 1;
```

```
for(i = 0; i < n; i++){
```

```
if((fichero = open(name_f[i],  
O_CREAT | O_WRONLY, 0644)) ==  
0){
```

```
perror("Open no pudo crear  
el directorio, Por favor  
verifiquelo.");
```

```
}
```

```
write(fichero, conte[i],
```

```
strlen(conte[i]));
```

```
close(fichero);
```

```
}
```

```
printf("\n\nEscriba el nombre  
del archivo: ");
```

```
scanf("%s", ruta);
```

```
setbuf(stdin, NULL);
```

```
printf("\n\nEscriba el modo en  
octal de los permisos: ");
```

```
scanf("%i", &modo_permisos);
```

```
setbuf(stdin, NULL);
```

```
if(chmod(ruta, modo_permisos)  
== 0)
```

```
printf("\n\nExito al cambiar los  
permisos.");
```

```
else
```

```
printf("\n\nError al cambiar los  
permisos.");
```

```
printf("\n\nPresione ENTER para  
continuar...");
```

```
getc(stdin);
```

```
setbuf(stdin, NULL);
```

```
return 0;
```

```
}
```

- **Capturas de pantalla**

```
Terminal
jjonatan@jjonatan-HP-14-Notebook-PC: ~/Escritorio/Practica4/Programa7
jjonatan@jjonatan-HP-14-Notebook-PC:~/Escritorio/Practica4/Programa7$ gcc -o Programa7 Programa7.c
jjonatan@jjonatan-HP-14-Notebook-PC:~/Escritorio/Practica4/Programa7$ ./Programa7
Soy el hijo ejecutando /home/jjonatan/Escritorio/Practica4/Programa7/arbol
Escribe la expresion en postfijo
12+
La expresion es:
1+2
El Valor de la expresion es: 3
SOY EL PAPA DEL PRIMER PROCESO
Soy el hijo 2 ejecutando /home/jjonatan/Escritorio/Practica4/Programa7/permisos
Escriba el nombre del archivo: archivo.txt
Escriba el modo en octal de los permisos: 8
Error al cambiar los permisos.
Presione ENTER para continuar...
SOY EL PADRE DEL SEGUNDO PROCESO
Soy el hijo 3 ejecutando /home/jjonatan/Escritorio/Practica4/Programa7/matriz_inversa
Sus respectivas inversas son:
Primera matriz:
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
Segunda matriz:
No tiene inversaa
SOY EL PADRE DEL TERCER PROCESO
SOY EL SUPER PADRE
jjonatan@jjonatan-HP-14-Notebook-PC:~/Escritorio/Practica4/Programa7$
```

Sección Windows

1.- Aguirre Cruz Eder Jonathan

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.

- **Código fuente**

```
#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 ( argc != 2 )
    {
```

```

        printf("Usar: %s\n",argv[0]);
        return 1;
    }

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

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

    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
    return 0;
}

```

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

```

#include <windows.h>
#include <stdio.h>
int main()
{
    printf("Soy el
hijo\n");
    exit(0);
}

```

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.

El primer programa, el proceso padre llama al hijo a través de la función CreateProcess()

- **Capturas de pantalla**

```

C:\Users\Juniocar1\Desktop\Practica 9>gcc hijo.c -o hijo
hijo.c:7:2: warning: no newline at end of file
C:\Users\Juniocar1\Desktop\Practica 9>gcc hijo.c -o hijo
C:\Users\Juniocar1\Desktop\Practica 9>gcc uno.c -o padre
C:\Users\Juniocar1\Desktop\Practica 9>padre hijo
Soy el padre
Soy el hijo
C:\Users\Juniocar1\Desktop\Practica 9>

```

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

A diferencia de sistemas operativos Linux, en Windows es un poco más complicada la tarea de crear un nuevo proceso, debido a la manera en la que hay que llamar al programa. De manera similar a Linux, existe la función WaitForSingleObject() que espera a que los nodos hijo terminen su tarea.

- 7.- Programe una aplicación que cree un proceso hijo a partir de un proceso padre, el hijo creado a su vez creara 10 hijos más. A su vez cada uno de los tres procesos creara 5 procesos más. Cada uno de los procesos imprimirá en pantalla su identificador.

- **Código fuente**

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;
}
```

10 Hijos

```

#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 < 10 ; j++)
    {

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

        if ( !CreateProcess(NULL, "dos", 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;
}

```

5 Hijos

```

#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 < 5 ; j++)
    {
        ZeroMemory(&si, sizeof(si));
        si.cb = sizeof(si);
        ZeroMemory(&pi, sizeof(pi));
    }
}

```

```

        if ( !CreateProcess(NULL, "final", 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;
}

```

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;
}

```


- Capturas de pantalla

```

**ARBOL DE PROCESOS**

->Soy el Padre, PID: 5064
  Nivel 2
  --->Soy el hijo, PID: 5752
    Nivel 3
    --->Soy el hijo, PID: 4744
      Nivel 4
      --->Soy el hijo, PID: 4724
      Nivel 4
      --->Soy el hijo, PID: 4224
      Nivel 4
      --->Soy el hijo, PID: 2652
      Nivel 4
      --->Soy el hijo, PID: 5224
      Nivel 4
      --->Soy el hijo, PID: 3976
    Nivel 3
    --->Soy el hijo, PID: 7140
      Nivel 4
      --->Soy el hijo, PID: 4712
      Nivel 4
      --->Soy el hijo, PID: 88
      Nivel 4
      --->Soy el hijo, PID: 4132
      Nivel 4
      --->Soy el hijo, PID: 2508
      Nivel 4
      --->Soy el hijo, PID: 5560
    Nivel 3
    --->Soy el hijo, PID: 5968
      Nivel 4
      --->Soy el hijo, PID: 6892
      Nivel 4
      --->Soy el hijo, PID: 6012
      Nivel 4
      --->Soy el hijo, PID: 184

```

```

  Nivel 4
  --->Soy el hijo, PID: 1008
  Nivel 4
  --->Soy el hijo, PID: 3600
  Nivel 3
  --->Soy el hijo, PID: 704
    Nivel 4
    --->Soy el hijo, PID: 6084
    Nivel 4
    --->Soy el hijo, PID: 1620
    Nivel 4
    --->Soy el hijo, PID: 1340
    Nivel 4
    --->Soy el hijo, PID: 2176
    Nivel 4
    --->Soy el hijo, PID: 3668
  Nivel 3
  --->Soy el hijo, PID: 4612
    Nivel 4
    --->Soy el hijo, PID: 2220
    Nivel 4
    --->Soy el hijo, PID: 4700
    Nivel 4
    --->Soy el hijo, PID: 2500
    Nivel 4
    --->Soy el hijo, PID: 916
    Nivel 4
    --->Soy el hijo, PID: 5868
  Nivel 3
  --->Soy el hijo, PID: 6428
    Nivel 4
    --->Soy el hijo, PID: 6080
    Nivel 4
    --->Soy el hijo, PID: 1196
    Nivel 4
    --->Soy el hijo, PID: 4424
    Nivel 4
    --->Soy el hijo, PID: 1424

```

2.- Buendía Moreno Hugo Vidal

8.- Programe la aplicación desarrollada en el punto 5 de la sección de Linux utilizando esta vez la creación de procesos en Windows

```

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

int main(){
    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    char p1[] = "suma";
    char p2[] = "resta";

```

```
char p3[] = "multiplicacion";
char p4[] = "transpuesta";
char p6[] = "lectura";
```

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

```
//Creación procesos hijos
```

```
if(!CreateProcess(NULL, p1, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)){
    printf("Fallo al invocar CreateProcess(%d)\n", GetLastError());
    return 0;
}
```

```
if(!CreateProcess(NULL, p2, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)){
    printf("Fallo al invocar CreateProcess(%d)\n", GetLastError());
    return 0;
}
```

```
if(!CreateProcess(NULL, p3, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)){
    printf("Fallo al invocar CreateProcess(%d)\n", GetLastError());
    return 0;
}
```

```
if(!CreateProcess(NULL, p4, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)){
    printf("Fallo al invocar CreateProcess(%d)\n", GetLastError());
    return 0;
}
```

```
if(!CreateProcess(NULL, p6, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)){
    printf("Fallo al invocar CreateProcess(%d)\n", GetLastError());
    return 0;
}
```

```
//Proceso padre
```

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

```
//Terminación controlada del proceso e hilo asociado de ejecución
```

```
CloseHandle(pi.hProcess);
CloseHandle(pi.hThread);
printf("\nProcesos terminados...");
```

```
}
```

```
Procesos hijos:
```

```
//Suma
```

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

```
int main(void){
```

```
    FILE *f1, *f2;
```

```
    int m1[15][15], m2[15][15];
```

```
    int i,j;
```

```
    char n;
```

```
    f1 = fopen("M1.txt","r");
```

```
    f2 = fopen("M2.txt","r");
```

```
    for(i=0; i<15; i++){
```

```
        for(j=0; j<15; j++){
```

```
            n = fgetc(f1);
```

```
            m1[i][j] = (int)(n - 48);
```

```

        n = fgetc(f2);
        m2[i][j] = (int)(n - 48);
        m1[i][j] += m2[i][j];
    }
}

f2 = fopen("suma.txt", "w");
fprintf(f2, "\nResultado de la suma:\n\n");
for(i=0; i<15; i++){
    fprintf(f2, " | ");
    for(j=0; j<15; j++){
        if(j==14)
            fprintf(f2, "%d", m1[i][j]);
        else
            fprintf(f2, "%d\t", m1[i][j]);
    }
    fprintf(f2, " | \n");
}
}

//Resta
#include<stdio.h>

int main(void){
    FILE *f1, *f2;
    int m1[15][15], m2[15][15];
    int i,j;
    char n;

    f1 = fopen("M1.txt", "r");
    f2 = fopen("M2.txt", "r");
    for(i=0; i<15; i++){
        for(j=0; j<15; j++){
            n = fgetc(f1);
            m1[i][j] = (int)(n - 48);
            n = fgetc(f2);
            m2[i][j] = (int)(n - 48);
            m1[i][j] -= m2[i][j];
        }
    }

    f2 = fopen("resta.txt", "w");
    fprintf(f2, "\nResultado de la resta:\n\n");
    for(i=0; i<15; i++){
        fprintf(f2, " | ");
        for(j=0; j<15; j++){
            if(j==14)
                fprintf(f2, "%d", m1[i][j]);
            else
                fprintf(f2, "%d\t", m1[i][j]);
        }
        fprintf(f2, " | \n");
    }
}

```

```

//Multiplicación
#include<stdio.h>

int main(void){
    FILE *f1, *f2;
    int m1[15][15], m2[15][15], mr[15][15];
    int i,j,k;
    char n;

    f1 = fopen("M1.txt","r");
    f2 = fopen("M2.txt","r");
    for(i=0; i<15; i++){
        for(j=0; j<15; j++){
            n = fgetc(f1);
            m1[i][j] = (int)(n - 48);
            n = fgetc(f2);
            m2[i][j] = (int)(n - 48);
        }
    }

    for(i=0; i<15; i++)
        for(j=0; j<15; j++)
            mr[i][j] = 0;

    for(i=0; i<15; i++)
        for(j=0; j<15; j++)
            for(k=0; k<15; k++)
                mr[i][j] += m1[i][k]*m2[k][j];

    f2 = fopen("multiplicacion.txt","w");
    fprintf(f2, "\nResultado de la multiplicacion:\n\n");
    for(i=0; i<15; i++){
        fprintf(f2, "| ");
        for(j=0; j<15; j++){
            if(j==14)
                fprintf(f2, "%d", mr[i][j]);
            else
                fprintf(f2, "%d\t", mr[i][j]);
        }
        fprintf(f2, " | \n");
    }
}

//Transpuesta
#include<stdio.h>
int main(void){
    FILE *f1, *f2;
    int m1[15][15], m2[15][15];
    int i,j;
    char n;
    f1 = fopen("M1.txt","r");
    f2 = fopen("M2.txt","r");
    for(i=0; i<15; i++){
        for(j=0; j<15; j++){
            n = fgetc(f1);
            m1[j][i] = (int)(n - 48);

```

```

        n = fgetc(f2);
        m2[j][i] = (int)(n - 48);
    }
}
f2 = fopen("transpuesta.txt","w");
fprintf(f2, "\nResultado de las transpuestas: \n\nM1: \n");
for(i=0; i<15; i++){
    fprintf(f2, " | ");
    for(j=0; j<15; j++){
        if(j==14)
            fprintf(f2, "%d", m1[i][j]);
        else
            fprintf(f2, "%d\t", m1[i][j]);
    }
    fprintf(f2, " | \n");
}
fprintf(f2, "\n\nM2: \n");
for(i=0; i<15; i++){
    fprintf(f2, " | ");
    for(j=0; j<15; j++){
        if(j==14)
            fprintf(f2, "%d", m1[i][j]);
        else
            fprintf(f2, "%d\t", m1[i][j]);
    }
    fprintf(f2, " | \n");
}
}

//Lectura
#include<stdio.h>
int main(void){
    FILE *fp;
    fp = fopen("suma.txt","r");
    while(!feof(fp)){
        printf("%c", fgetc(fp));
    }
    fp = fopen("resta.txt","r");
    while(!feof(fp)){
        printf("%c", fgetc(fp));
    }
    fp = fopen("multiplicacion.txt","r");
    while(!feof(fp)){
        printf("%c", fgetc(fp));
    }
    fp = fopen("transpuesta.txt","r");
    while(!feof(fp)){
        printf("%c", fgetc(fp));
    }
}

```

➤ Análisis crítico

- Aguirre Cruz Eder Jonathan

Esta práctica, la cual fue sencilla y fácil de realizar, abordo conceptos que se han estado viendo en clase cuando se entró a la segunda unidad: Administración de procesos. En mi opinión, desarrollar este tipo de prácticas es muy útil debido que ofrece la oportunidad de poner en practica la mayoría de los conceptos en clase, acción que en diversas materias no se realiza por falta de tiempo.

El comando *ps* ofrece la oportunidad de ver todos los procesos que se están ejecutando en una terminal, cosa que en este caso a mí me pareció un poco inútil (digo que en este caso, porque me imagino un caso en el que se pueda estar ejecutando una aplicación desarrollada en c que maneje procesos de manera continua y se necesite saber cuáles de esos procesos se están ejecutando y cuales no). El que me pareció más interesante de observar fue el comando con sus respectivos modificadores *-fea*, ya que ofrecía mayor información acerca de todos los procesos que se estaban ejecutando en el sistema operativo. Esto puede ser de gran utilidad para utilizar procesos que han llegado a estado zombie y se necesiten matar con el comando *kill*. Con *ps -fea* podremos saber el PID del proceso que se ha llegado a estado zombie y matarlo. El comando *top* me parece también muy interesante para verificar y monitorear los diversos procesos que se ejecutan en nuestro sistema operativo. Puede de resultar de gran ayuda para ver si algún usuario está haciendo mal uso de este y poder controlarlo o contenerlo.

Hablando de los programas de prueba, opino que fue de gran ilustración para comprender la creación de un proceso por copia exacta con ayuda de la función *fork()*, que se encuentra en la Librería *<unistd.h>*. Solo fueron aplicaciones que imprimían diferentes cadenas dependiendo si era un hijo o un padre.

El primer programa a realizar, se pudo aplicar el uso de estructuras repetitivas para la creación de procesos que requieran un número grande de procesos, ya sean por niveles, o por el mismo nivel. Las funciones *getpid()* y *getppid()* fueron de gran utilidad debido a que se debía conocer e imprimir el PID de los distintos hijos y el PID del padre.

- **Buendía Moreno Hugo Vidal**

Es importante mencionar que algunos de los comandos necesarios para la práctica eran totalmente desconocidos, por lo que se investigó, con ayuda del manual de Linux y de internet, la utilización y función de dichos comandos, como *chmod*, *stat*, *cp*, *access*, *fcntl*, entre otros. En cada uno de los comandos, se indicaban las librerías que es necesario utilizar para implementar aplicaciones en C, el cual es nuestro caso; esto nos fue de mucha ayuda pues facilitó la implementación de ellas en el programa.

En mi opinión, el análisis que requería el programa era complejo, pues ya se deben tener conocimientos previos del sistema operativo con el que se está trabajando y la forma en que manipula los archivos, permisos, rutas, entre otros.

No fue necesario instalar la memoria desde consola, pues el sistema la reconoció automáticamente. Aquí se puede observar que se necesitan permisos especiales de súper usuario para realizar cambios, pues es una nueva unidad a la que se tendrá acceso y que a la vez tendrá acceso a los elementos del sistema.

Con esta práctica se pueden observar las múltiples opciones de manejo de datos que nos ofrece Linux, que en realidad son bastante útiles y amplían la visión de lo que se está trabajando; en el caso de Windows, personalmente, no conozco algunos comandos que sean equivalentes a los tratados aquí, como por ejemplo, la modificación de los permisos por parte directamente del usuario para acceder a algún archivo indicando la ruta correspondiente.

- **Saules Cortés Jhonatan**

Me parecieron bastante útiles algunos comandos que se revisaron en la práctica, que nos ayudan a conocer un poco más sobre el funcionamiento del sistema operativo, los procesos que se ejecutan en el mismo, y aumentar el espacio en el procesador para ejecutar procesos útiles o necesarios en lugar de alguno que esté ocioso y sin alguna utilidad, con la información siguiente:

- Identificación del proceso.
- Identificación del proceso padre.
- Información sobre el usuario y grupo.
- Estado del procesador.
- Información de control de proceso
- Información del planificador.
- Segmentos de memoria asignados.
- Recursos asignados.

Con la realización del primer programa hubo un poco de confusión para crear los ciclos de creación de los procesos hijos, en forma horizontal y vertical, pero se solucionó revisando paso por paso el funcionamiento del ciclo y aumentando la cantidad de procesos a medida que avanzaba el programa, me pareció que se aprendió mejor cómo es realmente la creación de una cantidad grande de procesos, indicando el id del mismo y de su proceso padre.

Se llevó a la práctica lo que se vio teóricamente y se comprobó que en teoría parece un poco más sencillo que cuando se lleva a cabo, que es cuando realmente surgen las dudas pero al mismo tiempo se entiende mejor la estructura, en este caso, de los árboles de procesos.

Dichos árboles me parecen de gran utilidad, pues cada uno puede realizar operaciones independientes o codependientes, según se elija, pues también se tiene

esa posibilidad de compartir las operaciones realizadas o aislarlas entre padres e hijos, por medio de la manipulación de la instrucción `exit(0)`.

➤ Conclusiones

- **Aguirre Cruz Eder Jonathan**

Podemos decir que esta práctica fue de gran importancia por los siguientes puntos:

- ✓ Ver un comando que nos ofrece la oportunidad de ver que procesos se ejecutan en nuestro sistema operativo. Los modificadores del comando pueden ayudarnos a realizar acciones muy específicas que nosotros podríamos requerir.
- ✓ Ver de manera práctica (con ayuda del gran sistema operativo Linux) como se pueden crear procesos por copia exacta de código a través de la llamada a la función en lenguaje c *fork()*.
- ✓ Se ocuparon las funciones de utilidad para procesos que nos ofrece linux, las cuales son: `getpid()`, `getppid()` y `wait()`.
- ✓ Crear un árbol de procesos propuesto. El que se desarrolló en el primer programa propuesto solo se creó la estructura, es decir, se creó sin ninguna instrucción (instrucción que desarrolle alguna tarea específica) dentro del padre o hijos y se podría decir que es inútil, pero puede ser de gran ayuda si se ocupa para un fin en específico que requiere ese tipo de árbol de procesos. Ya en el segundo programa se pudo observar como un árbol con 6 hijos puede desarrollar diversas tareas, que en este caso fue operaciones con matrices (suma, restar, multiplicar, sacar sus respectivas traspuestas inversas).
- ✓ Se pudo comprender superficialmente el diagrama de estados de cada proceso. Se dice superficialmente porque no se vio de manera explícita como se cambiaba de estado y en qué momento, si no se vio en el segundo programa a desarrollar como los hijos puede trabajar de manera concurrente, y el padre entra en un estado de espera para que los hijos puedan terminar las tareas asignadas, para si los hijos no queden huérfanos.

Para concluir: Esta práctica si cumplió con la competencia estipulada, la cual fue que el alumno aprenderá a familiarizarse con el administrador de procesos del sistema operativo Linux, a través de la creación de nuevos procesos, ejecución y terminación. Se pudo comprender como se crean nuevos procesos, ejecutarlos en consola para realizar tareas específicas, y dar diversos estados a los procesos para acoplarnos a la tarea requerida.

- **Buendía Moreno Hugo Vidal**

Al término de esta práctica, se pudieron concretar algunas ideas sobre el sistema operativo Linux; en lo personal, sólo conocía a grandes rasgos el sistema, con las actividades realizadas, fue posible aprender un poco más sobre las funcionalidades que ofrece Linux. En cuanto a los formatos del texto procesados en la práctica, ya se observó que hay gran variedad de éstos, depende de los propósitos que se tengan, por ejemplo, para realizar programas es conveniente utilizar un editor de texto sencillo para lograr la unificación del formato del código y no existan problemas al ejecutarlo o reutilizarlo.

Las codificaciones son un tema complejo, lo cual se comprobó con la modificación del archivo de Word, que impidió que posteriormente el sistema lo abriera correctamente. La práctica sirvió para familiarizarse más con el sistema operativo Linux, que es con el que se estará trabajando a lo largo de las prácticas, y que, por supuesto, se irán conociendo nuevas modalidades y funciones del sistema con el avance de las mismas. Finalmente, creo que se cumplió con los objetivos establecidos al principio de la práctica, he aprendido un poco más sobre este sistema operativo, y creo que tiene más funcionalidades, en nuestro caso, que el sistema Windows. Puede ser que ofrezca un entorno de trabajo más amplio en cuanto a la programación y la implementación de funciones, que nos facilitan la elaboración de proyectos con instrucciones o procesos un poco más complejos.

- **Saules Cortés Jhonatan**

Después de realizar esta práctica se aprendieron nuevos comandos del sistema operativo Linux, para administrar los procesos que se ejecutan en nuestra computadora. Por otra parte, la administración de procesos en esta distribución nos permite crear diferentes procesos por medio de la copia exacta de código, con lo cual se hace justamente la copia del código del proceso padre en ese momento, y comienza a ejecutarse desde la parte del código donde aparece la instrucción `fork()`.

Es importante considerar el tiempo de duración de la ejecución de los procesos, por ejemplo en el primer programa, para evitar que el proceso padre termine antes que los procesos hijos terminen de ejecutarse, y que no haya algún problema con el identificador del proceso, pues en ese caso, se redireccionaría hasta el primer proceso en funcionamiento en nuestro procesador y ya no se estaría cumpliendo la función que se requería al principio. Para realizar esto se utilizó la instrucción `wait` en algunas partes del código antes de crear los procesos hijos, contemplando el tiempo en que se ejecuta cada uno de ellos.

Con la realización del segundo programa se pudo observar una mayor utilidad a la aplicación de la administración de procesos, pues se dividieron las operaciones a realizar con las matrices, cada proceso tenía su función independiente y mostraba resultados al usuario.

Así, se conoció otra pequeña parte del sistema operativo de Linux, que cada vez nos muestra muchas herramientas útiles, especialmente para la programación y la planificación de los procesos. También cabe mencionar que se debe ser cuidadoso al crear nuevos procesos hijos, pues si no se colocan correctamente los `exit(0)`, el programa hará cosas inesperadas y creará procesos donde no los hay, o cambiaría totalmente la estructura a la que se tiene planeada inicialmente. Para concluir, esta práctica me agradó porque encontramos una herramienta útil para estructurar de una manera diferente los programas que desarrollemos posteriormente, cuando sea aplicable a la resolución del problema y nos ayude a optimizar el funcionamiento del mismo.