

# GUION 2

1) Bájate el programa “errores.c” y pruébalo. Verás que salen avisos. Haz man de man y observa las secciones que puedes ver en el manual. Observa sobre todo las secciones dos y tres. Comprueba con ldd con que librerías dinámicas (compartidas) has enlazado (no confundir con ld que es el linker que se llama desde gcc). Identifica cuál es una de ellas.

Al ejecutar ldd a.out, se ve que aparece la librería dinámica libc.so

2) Prueba el programa de los errores para saber el número y mensajes de error de tu sistema. Cámbialo para usar stderr. Introduce una línea con perror y prueba por que canal sale el mensaje.

Con stderr sale por la línea 0 mientras que con perror sale por la línea 2, que es la de errores.

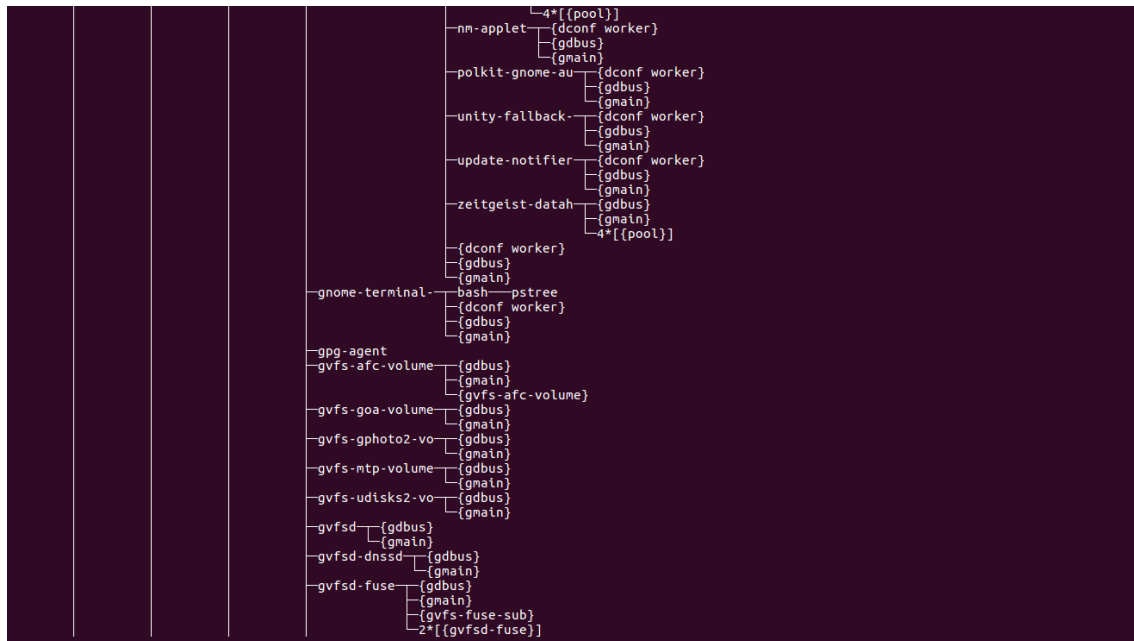
3) Utiliza el comando readelf con la opción -h y -S aplicado a un ejecutable que contenga un array global, una variable inicializada, y variables locales a main, Identifica al menos tres secciones que se han visto en teoría. ¿Si comentas las variables cambian los tamaños de los segmentos? Usa el comando objdump con la opción -a y -h de la misma manera. Los segmentos de stack o heap ¿aparecen en las secciones de ELF?

Se ven el text, data y el bss. Al comentar se modifica el .text (archivos drop).

Los segmentos stack y heap no se almacenan en el elf, si no que se generan en tiempo de ejecución.

4) Ejecuta el comando pstree y observa tu jerarquía de procesos. Indica dónde aparece el propio pstree y de quien cuelga. Retrocede hasta llegar al proceso inicial del sistema y escribe por todos los que se pasan para llegar a que ejecutes algún comando. Ahora entra desde un terminal virtual de texto y haz lo mismo (o si tienes la bash de W10) ¿cuál es la gran diferencia?

El pstree esta en : systemd/ligthdm/ligthdm/upstar/gnome-terminal/bash/pstree



La diferencia es que en el de la maquina virtual es mucho mas pequeño que el normal.

5) Vete al directorio /proc y lista su contenido. Identifica el PID de la Shell, vete a su directorio y lista su contenido. Usa cat o ls para averiguar que contienen los ficheros que aparecen. Localiza (debes saber ya el comando gracias al guion 1) en que fichero está el parámetro vruntime, y averigua para que se usa. Si lo buscas en la bash de W10 no existe, ¿por qué?

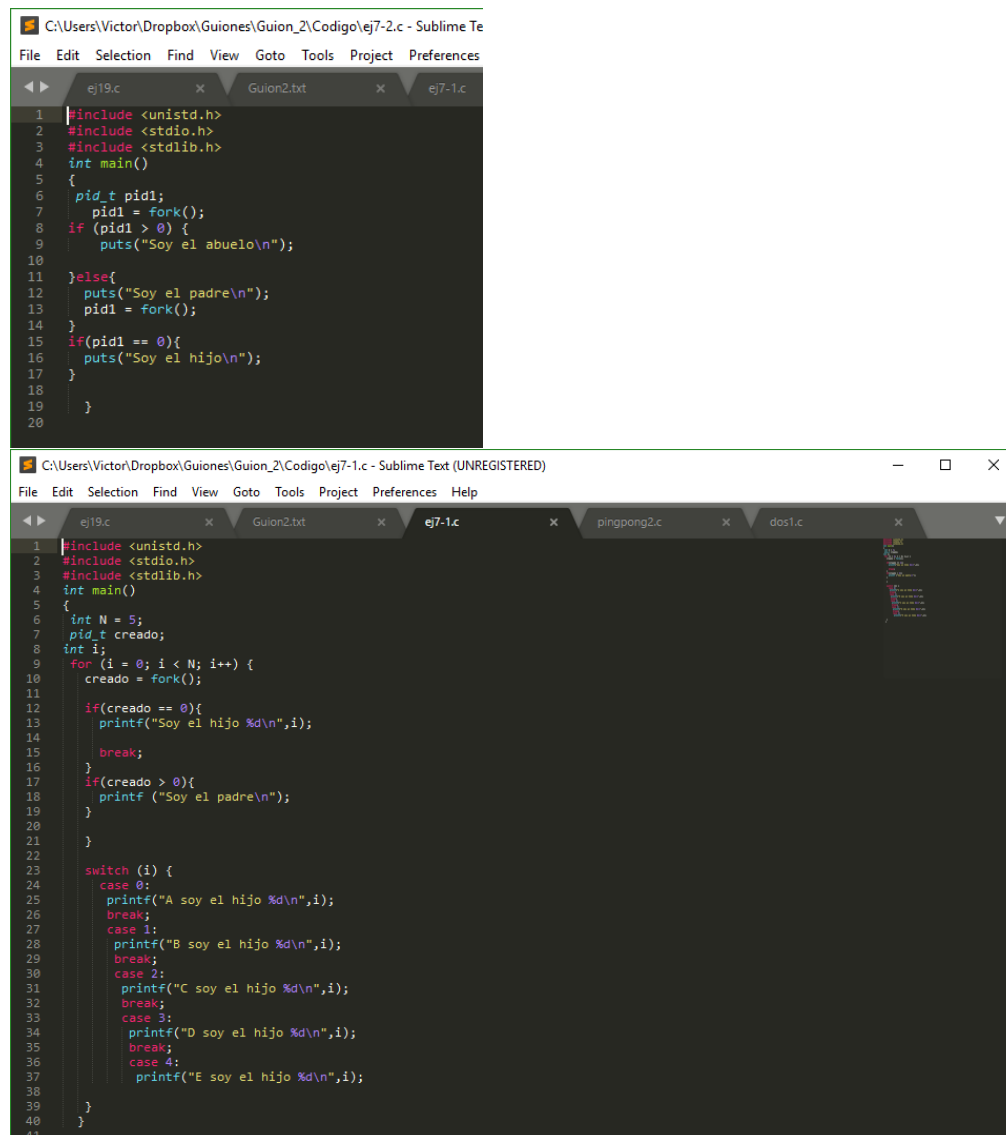
```
victor@victor-Lenovo-G50-80:/proc$ ps
  PID TTY          TIME CMD
  5536 pts/18    00:00:00 bash
  6854 pts/18    00:00:00 ps
victor@victor-Lenovo-G50-80:/proc$ ls 5536/
attr          environ      mem          pagemap      stack
autogroup     exe          mountinfo   patch_state  stat
auxv          fd           mounts      personality  statm
cgroup        fdinfo       mountstats  projid_map   status
clear_refs    gid_map     net         root         syscall
cmdline       io          ns          sched         task
conn          limits      numa_maps   schedstat    timers
coredump_filter lognuid      oom_adj     sessionid    timerslack_ns
cpuset        map_files   oom_score   setgroups    uid_map
cwd           maps        oom_score_adj snaps         wchan
victor@victor-Lenovo-G50-80:/proc$
```

grep -r vruntime, este comando muestra el tiempo que se ha estado ejecutando la CPU.

6) Obtén este código (ejem\_fork.c) de la página, entiéndelo y pruébalo. Date cuenta cómo aparecen dos mensajes de los dos procesos e indica por qué canal saldría el mensaje de error. Si desvías la salida a un fichero ¿qué ocurre?

El mensaje de error saldría por el canal 2 ya que se usa perror.

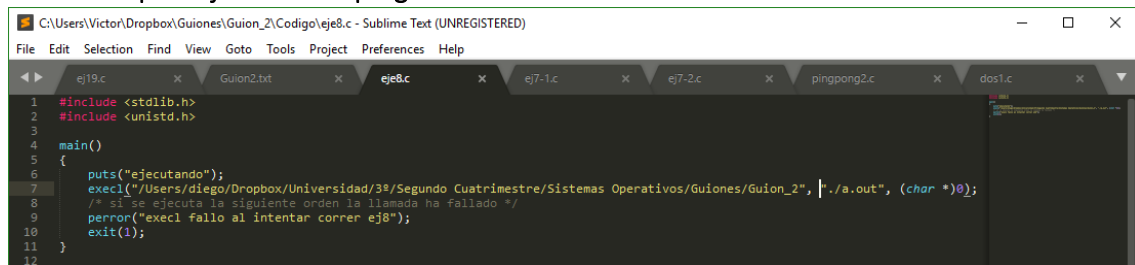
7) Realiza un programa en que un proceso padre genera N procesos hijo iguales (hermanos entre ellos). Crea otro programa que contenga tres generaciones de procesos: Abuelo padre y nieto.



```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 int main()
5 {
6     pid_t pid1;
7     pid1 = fork();
8     if (pid1 > 0) {
9         puts("Soy el abuelo\n");
10    }
11    else{
12        puts("Soy el padre\n");
13        pid1 = fork();
14    }
15    if(pid1 == 0){
16        puts("Soy el hijo\n");
17    }
18 }
19
20
```

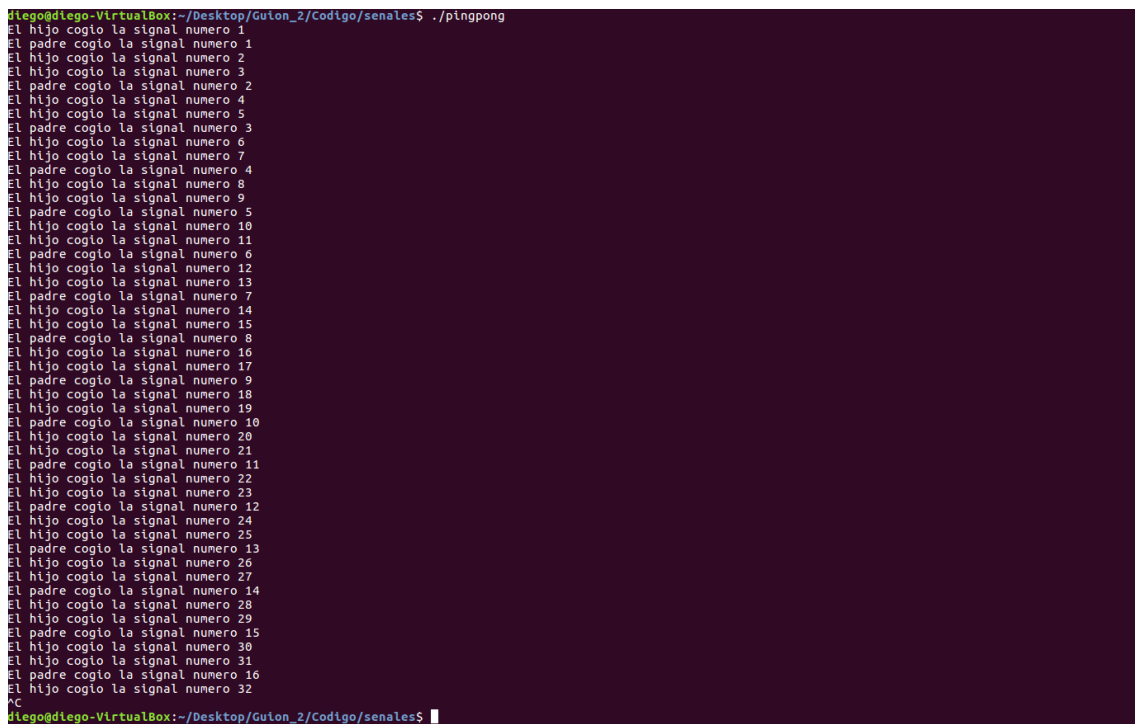
```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 int main()
5 {
6     int N = 5;
7     pid_t creado;
8     int i;
9     for (i = 0; i < N; i++) {
10        creado = fork();
11
12        if(creado == 0){
13            printf("Soy el hijo %d\n",i);
14            break;
15        }
16        if(creado > 0){
17            printf ("Soy el padre\n");
18        }
19    }
20
21    switch (i) {
22    case 0:
23        printf("A soy el hijo %d\n",i);
24        break;
25    case 1:
26        printf("B soy el hijo %d\n",i);
27        break;
28    case 2:
29        printf("C soy el hijo %d\n",i);
30        break;
31    case 3:
32        printf("D soy el hijo %d\n",i);
33        break;
34    case 4:
35        printf("E soy el hijo %d\n",i);
36    }
37 }
38
39
40
41
```

8) Descárgate los programas (ejem\_exec1.c y 2) de la página, entiéndelos, compílalos y ejecútalos para que compruebes como desde tu propio programa se ejecuta un comando del sistema. Modifícalo para ejecutar otro programa a tu elección.



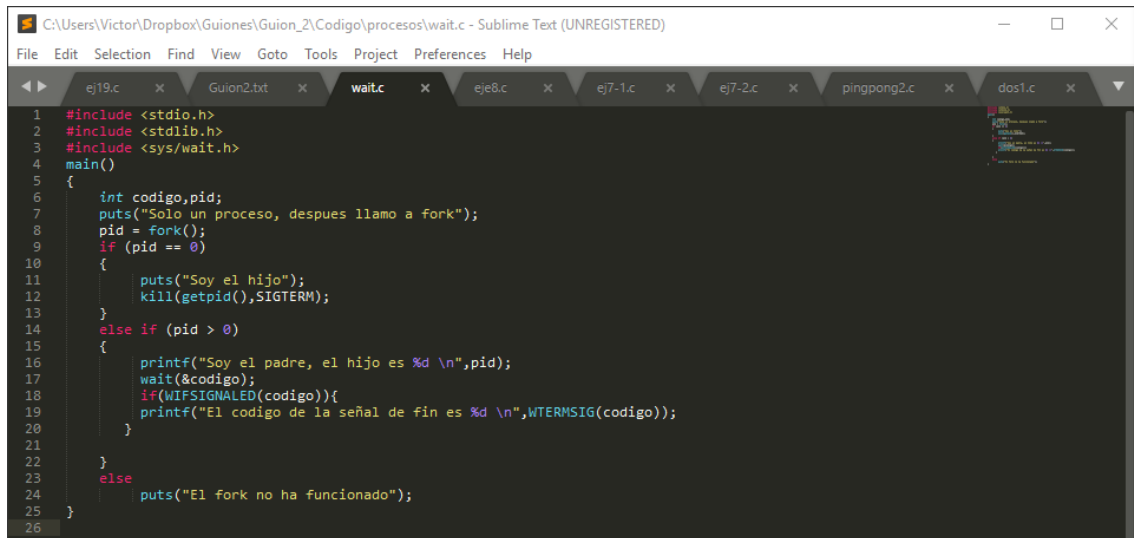
```
1 #include <stdlib.h>
2 #include <unistd.h>
3
4 main()
5 {
6     puts("ejecutando");
7     execl("/Users/diego/Dropbox/Universidad/3º/Segundo Cuatrimestre/Sistemas Operativos/Guiones/Guion_2", |./a.out", (char *)0);
8     /* si se ejecuta la siguiente orden la llamada ha fallado */
9     perror("execl fallo al intentar correr ej8");
10    exit(1);
11 }
12
```

9) Descárgate el programa de la página (param.c), entiéndelo, compílalolo y ejecútalolo con un argumento (variable) que pertenezca al entorno del programa. Si no conoces ninguno prueba con cualquiera (printenv) y fíjate lo que sale.



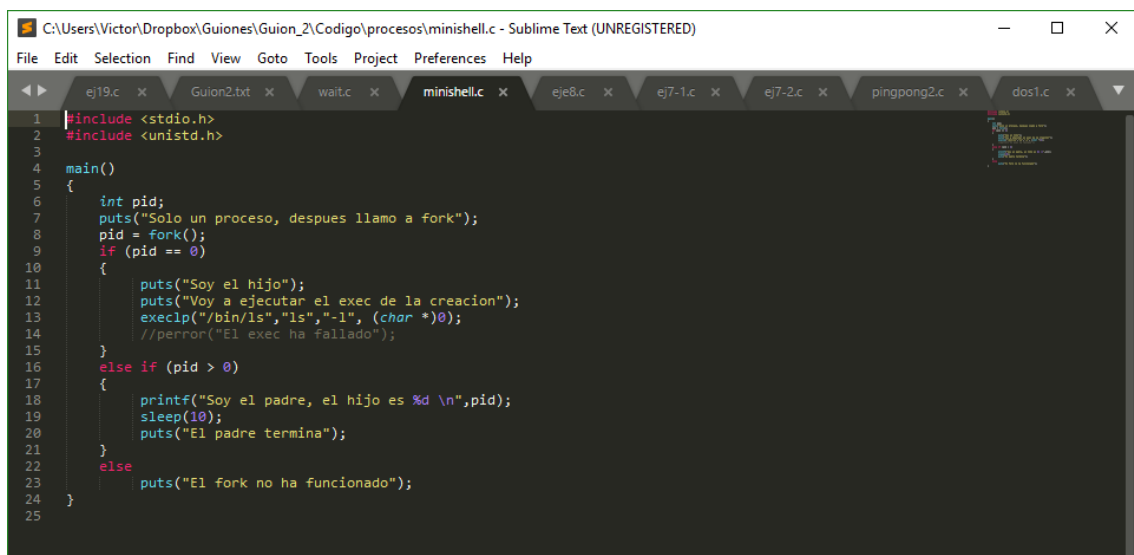
```
Diego@diego-VirtualBox:~/Desktop/Guion_2/Codigo/senales$ ./pingpong
El hijo cogio la signal numero 1
El padre cogio la signal numero 1
El hijo cogio la signal numero 2
El hijo cogio la signal numero 3
El padre cogio la signal numero 2
El hijo cogio la signal numero 4
El hijo cogio la signal numero 5
El padre cogio la signal numero 3
El hijo cogio la signal numero 6
El hijo cogio la signal numero 7
El padre cogio la signal numero 4
El hijo cogio la signal numero 8
El hijo cogio la signal numero 9
El padre cogio la signal numero 5
El hijo cogio la signal numero 10
El hijo cogio la signal numero 11
El padre cogio la signal numero 6
El hijo cogio la signal numero 12
El hijo cogio la signal numero 13
El padre cogio la signal numero 7
El hijo cogio la signal numero 14
El hijo cogio la signal numero 15
El padre cogio la signal numero 8
El hijo cogio la signal numero 16
El hijo cogio la signal numero 17
El padre cogio la signal numero 9
El hijo cogio la signal numero 18
El hijo cogio la signal numero 19
El padre cogio la signal numero 10
El hijo cogio la signal numero 20
El hijo cogio la signal numero 21
El padre cogio la signal numero 11
El hijo cogio la signal numero 22
El hijo cogio la signal numero 23
El padre cogio la signal numero 12
El hijo cogio la signal numero 24
El hijo cogio la signal numero 25
El padre cogio la signal numero 13
El hijo cogio la signal numero 26
El hijo cogio la signal numero 27
El padre cogio la signal numero 14
El hijo cogio la signal numero 28
El hijo cogio la signal numero 29
El padre cogio la signal numero 15
El hijo cogio la signal numero 30
El hijo cogio la signal numero 31
El padre cogio la signal numero 16
El hijo cogio la signal numero 32
^C
Diego@diego-VirtualBox:~/Desktop/Guion_2/Codigo/senales$
```

10) Descárgate el programa de la página (wait.c), entiéndelo, compílalo y ejecútalo. Indica para qué sirven las macros del programa. Edita el programa para hacer uso de las macros dentro de sentencias if. Usa también de WIFSIGNALED y WTERMSIG que vienen descritas en el manual.



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/wait.h>
4 main()
5 {
6     int codigo,pid;
7     puts("Solo un proceso, despues llamo a fork");
8     pid = fork();
9     if (pid == 0)
10    {
11        puts("Soy el hijo");
12        kill(getpid(),SIGTERM);
13    }
14    else if (pid > 0)
15    {
16        printf("Soy el padre, el hijo es %d \n",pid);
17        wait(&codigo);
18        if(WIFSIGNALED(codigo)){
19            printf("El codigo de la señal de fin es %d \n",WTERMSIG(codigo));
20        }
21    }
22    else
23        puts("El fork no ha funcionado");
24 }
25
26
```

11) Descárgate el programa minishell.c de la página, entiéndelo, compílalo y prueba su funcionamiento. Comenta la línea del wait y observa que pasa. Realiza cambios en el programa para que veas la existencia de zombies, recuerda cómo se ejecutan procesos en background y que la llamada a sleep duerme a un proceso varios segundos que son puestos como argumento.



```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 main()
5 {
6     int pid;
7     puts("Solo un proceso, despues llamo a fork");
8     pid = fork();
9     if (pid == 0)
10    {
11        puts("Soy el hijo");
12        puts("Voy a ejecutar el exec de la creacion");
13        execlp("/bin/ls","ls","-l", (char *)0);
14        //perror("El exec ha fallado");
15    }
16    else if (pid > 0)
17    {
18        printf("Soy el padre, el hijo es %d \n",pid);
19        sleep(10);
20        puts("El padre termina");
21    }
22    else
23        puts("El fork no ha funcionado");
24 }
25
```

12) Descárgate el programa datos.c de la página, entiéndelo, compílalo y prueba su funcionamiento. Comprueba el valor de la variable común en ambos procesos.

Al crear el proceso se copia la memoria, pero no se comparte.

13) Descárgate el programa fichero.c de la página, entiéndelo, compílalo y prueba su funcionamiento. Muestra el fichero creado en pantalla. Crea una copia y cambia el programa para que ahora tanto padre como hijo abran el fichero después del fork. Ejecútalo y comprueba de nuevo el contenido del fichero. ¿cuál es la diferencia?

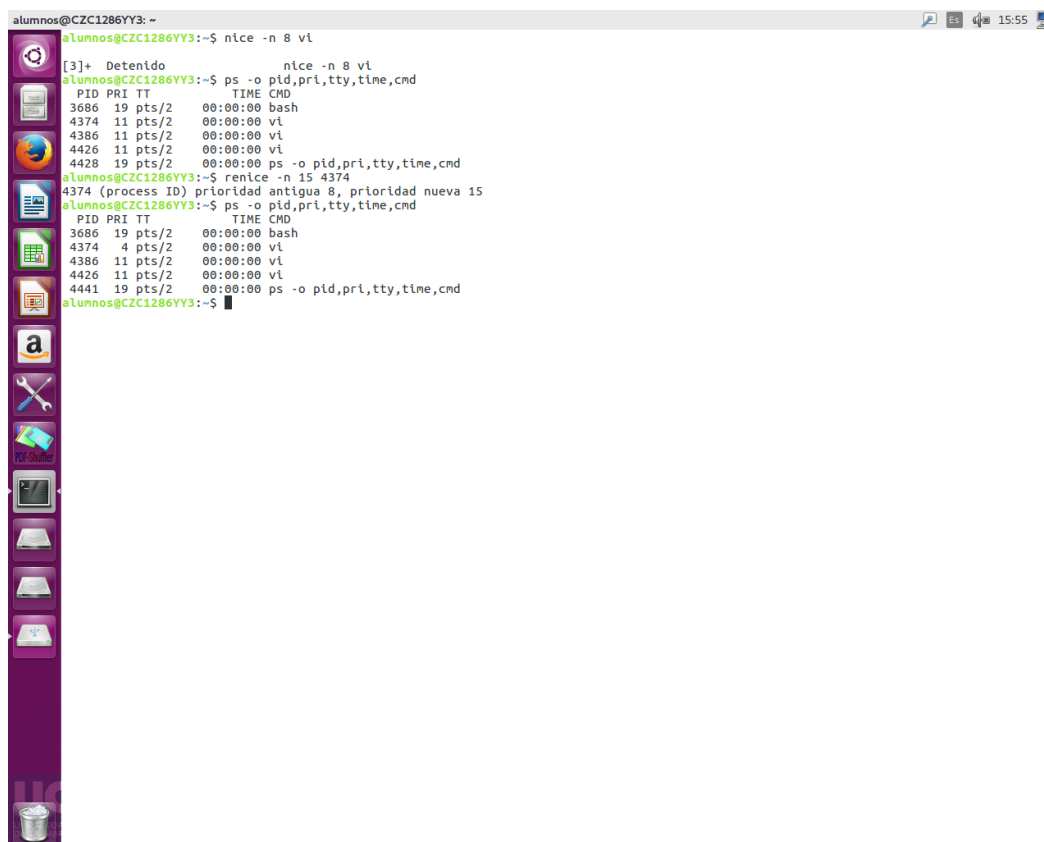
El padre se escribe mas tarde porque es el ultimo en ejecutarse, ya que tiene un sleep.

Si el fichero se ha abierto antes del fork se comparte, pero si se abre después se sobrescribe.

14) Descárgate el programa identifi.c de la página, entiéndelo, compílalo y prueba su funcionamiento. Indica los resultados que aparecen en pantalla e interprétalos.

Sale el PID del padre y del hijo, después el PPID de cada uno, el del padre es 1707 y el del hijo siempre es 1. Luego el grupo de cada uno, que es el mismo para ambos pero varia con cada ejecución.

15) Ejecuta el vi en background. A continuación, ejecútalo anteponiendo nice con un valor determinado. Haz un ps con la opción adecuada para mostrar las prioridades de los procesos que has ejecutado. Indica si has cambiado la prioridad del proceso. Intenta hacerlo con un valor negativo en un puesto fijo y en un portátil o Shell de W10. Observa la identificación del primer vi y cámbialo su prioridad. Compruébalo. ¿Es coincidente con lo que se mencionó en teoría?



```
alumnos@CZC1286YY3: ~  
alumnos@CZC1286YY3:~$ nice -n 8 vi  
[3]+ Detenido      nice -n 8 vi  
alumnos@CZC1286YY3:~$ ps -o pid,pri,TTY,time,cmd  
    PID PRI TT      TIME CMD  
    3686 19 pts/2    00:00:00 bash  
    4374 11 pts/2    00:00:00 vi  
    4386 11 pts/2    00:00:00 vi  
    4426 11 pts/2    00:00:00 vi  
    4428 19 pts/2    00:00:00 ps -o pid,pri,TTY,time,cmd  
alumnos@CZC1286YY3:~$ renice -n 15 4374  
4374 (process ID) prioridad antigua 8, prioridad nueva 15  
alumnos@CZC1286YY3:~$ ps -o pid,pri,TTY,time,cmd  
    PID PRI TT      TIME CMD  
    3686 19 pts/2    00:00:00 bash  
    4374  4 pts/2    00:00:00 vi  
    4386 11 pts/2    00:00:00 vi  
    4426 11 pts/2    00:00:00 vi  
    4441 19 pts/2    00:00:00 ps -o pid,pri,TTY,time,cmd  
alumnos@CZC1286YY3:~$
```

No, no deja usar un valor negativo.

16) Descárgate el programa anterior (prioridad.c) y Pruébalo. Ejecuta el comando top y el w y descubre y analiza los datos reflejados en pantalla. Ejecuta los comandos free y uptime , advierte las coincidencias. ¿de dónde cogen estos comandos la información que presentan?

top sirve para ver todos los procesos activos en el ordenador.

w muestra la información sobre los usuarios que están conectados en ese momento a la maquina y sobre sus procesos.

free muestra información sobre los usuarios que están conectados en ese momento a la maquina y sobre sus procesos.

uptime muestra información sobre el tiempo que lleva sin ser reiniciado el ordenador.

17) Descubre que tipo de arquitectura tienes en el sistema. De acuerdo con esto, ejecuta un editor vi y páralo. Mira su PID. Indica su afinidad en modo máscara y lista. Cambia su afinidad en los dos modos y compruébalo. Por último, lanza otro vi en los dos modos y con diferentes cores y compruébalo.

Con lscpu vemos que la arquitectura de mi sistema es x86-64.

```
victor@victor-Lenovo-G50-80: ~
victor@victor-Lenovo-G50-80:~$ vi
[1]+  Detenido          vi
victor@victor-Lenovo-G50-80:~$ ps
  PID TTY          TIME CMD
 2673 pts/4    00:00:00 bash
 3172 pts/4    00:00:00 vi
 3173 pts/4    00:00:00 ps
victor@victor-Lenovo-G50-80:~$ taskset -p 3172
pid 3172's current affinity mask: f
victor@victor-Lenovo-G50-80:~$ taskset -cp 3172
pid 3172's current affinity list: 0-3
victor@victor-Lenovo-G50-80:~$ taskset -p 0x1 3172
pid 3172's current affinity mask: f
pid 3172's new affinity mask: 1
victor@victor-Lenovo-G50-80:~$ taskset -cp 0 3172
pid 3172's current affinity list: 0
pid 3172's new affinity list: 0
victor@victor-Lenovo-G50-80:~$
```

18) Busca las señales que tienes en tu sistema, su numeración y el total disponible. Si no existe ese camino para `signal.h` averigua dónde está, usa lo aprendido en el guion 1. ¿cuáles crees que has usado?

64 señales, con el `kill -l` veo todas.

```
victor@victor-Lenovo-G50-80: ~  
victor@victor-Lenovo-G50-80:~$ kill -l  
1) SIGHUP      2) SIGINT      3) SIGQUIT      4) SIGILL      5) SIGTRAP  
6) SIGABRT     7) SIGBUS      8) SIGFPE       9) SIGKILL     10) SIGUSR1  
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE     14) SIGALRM     15) SIGTERM  
16) SIGSTKFLT  17) SIGCHLD   18) SIGCONT     19) SIGSTOP    20) SIGTSTP  
21) SIGTTIN    22) SIGTTOU   23) SIGURG      24) SIGXCPU    25) SIGXFSZ  
26) SIGVTALRM  27) SIGPROF   28) SIGWINCH    29) SIGIO       30) SIGPWR  
31) SIGSYS     34) SIGRTMIN   35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8  
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13  
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12  
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2  
63) SIGRTMAX-1 64) SIGRTMAX  
victor@victor-Lenovo-G50-80:~$
```

//captura

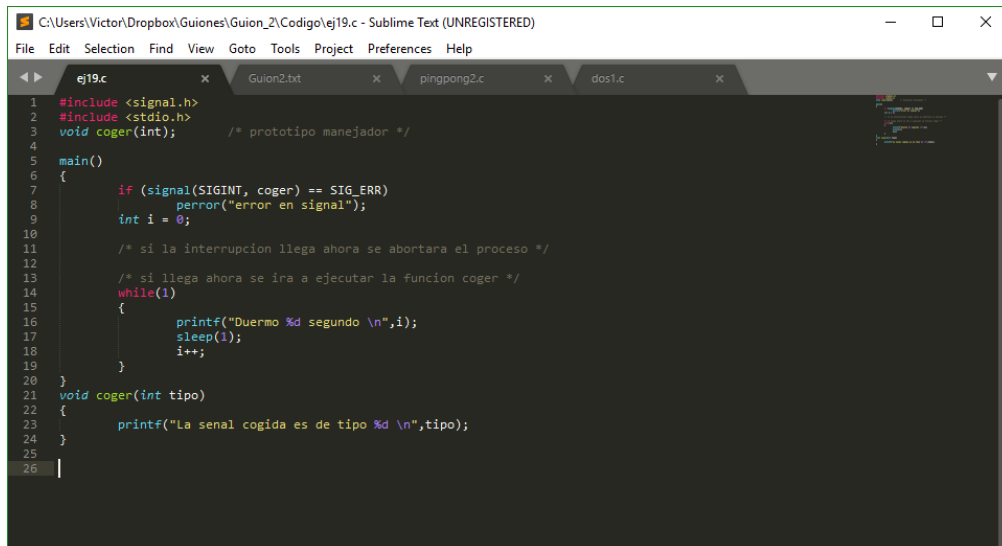
Hemos usado la señal `kill`.



19) Descárgate el programa controlc.c de la página, entiéndelo, compílalo y prueba su funcionamiento. Crea uno nuevo poniendo un sleep antes de signal y pulsa control-c para que veas que se aborta. Vuelve al original e indica cuántas veces puedes pulsar control-c antes de que acabe. ¿es más de una? Modifica el programa para que puedas pulsar varias veces control-c en cualquier sistema (por ejemplo, uno que no rearme el manejador) y que además la función de atención no pueda ser llamada dentro de la propia función de atención para que no se aniden.

El original se puede pulsar 4 veces antes de que se acabe.

Modificación:



```
1 #include <signal.h>
2 #include <stdio.h>
3 void coger(int); /* prototipo manejador */
4
5 main()
6 {
7     if (signal(SIGINT, coger) == SIG_ERR)
8         perror("error en signal");
9     int i = 0;
10
11     /* si la interrupcion llega ahora se abortara el proceso */
12
13     /* si llega ahora se ira a ejecutar la funcion coger */
14     while(1)
15     {
16         printf("Duermo %d segundo \n",i);
17         sleep(1);
18         i++;
19     }
20 }
21 void coger(int tipo)
22 {
23     printf("La senal cogida es de tipo %d \n",tipo);
24 }
25
26 |
```

20) Descárgate el programa pingpong.c de la página, entiéndelo, compílalo y prueba su funcionamiento. Cambia el programa (comenta una línea) para que los dos procesos tengan la misma función de atención. Cambia el programa anterior y comenta las líneas donde están los pauses. Compila y ejecuta. Indica qué ocurre.

```
Diego@diego-VirtualBox:~/Desktop/Guion_2/Codigo/senales$ ./pingpong
El hijo cogio la signal numero 1
El padre cogio la signal numero 1
El hijo cogio la signal numero 2
El hijo cogio la signal numero 3
El padre cogio la signal numero 2
El hijo cogio la signal numero 4
El hijo cogio la signal numero 5
El padre cogio la signal numero 3
El hijo cogio la signal numero 6
El hijo cogio la signal numero 7
El padre cogio la signal numero 4
El hijo cogio la signal numero 8
El hijo cogio la signal numero 9
El padre cogio la signal numero 5
El hijo cogio la signal numero 10
El hijo cogio la signal numero 11
El padre cogio la signal numero 6
El hijo cogio la signal numero 12
El hijo cogio la signal numero 13
El padre cogio la signal numero 7
El hijo cogio la signal numero 14
El hijo cogio la signal numero 15
El padre cogio la signal numero 8
El hijo cogio la signal numero 16
El hijo cogio la signal numero 17
El padre cogio la signal numero 9
El hijo cogio la signal numero 18
El hijo cogio la signal numero 19
El padre cogio la signal numero 10
El hijo cogio la signal numero 20
El hijo cogio la signal numero 21
El padre cogio la signal numero 11
El hijo cogio la signal numero 22
El hijo cogio la signal numero 23
El padre cogio la signal numero 12
El hijo cogio la signal numero 24
El hijo cogio la signal numero 25
El padre cogio la signal numero 13
El hijo cogio la signal numero 26
El hijo cogio la signal numero 27
El padre cogio la signal numero 14
El hijo cogio la signal numero 28
El hijo cogio la signal numero 29
El padre cogio la signal numero 15
El hijo cogio la signal numero 30
El hijo cogio la signal numero 31
El padre cogio la signal numero 16
El hijo cogio la signal numero 32
^C
```

Se ha comentado la línea en el caso del hijo, la de signal(SIGUSR1, hijo);

No se ejecutan alternándose el padre y el hijo, si no que se ejecutan al azar.

21) Crea una copia del original y haz que el proceso hijo retome la función anterior que tenía al cabo de 5 vueltas (para ello debes usar lo que devuelve signal y guardarlo en una variable de tipo puntero a función como este prototipo: void (\*antiguo)(int)). Cambia el programa y al cabo de 10 vueltas haz que recupere la acción por defecto y descubre qué le ocurre.

```
diego@diego-VirtualBox:~/Desktop/Guion_2/Codigo/senales$ ./pingpong2
El padre cogio la signal numero 1
El hijo cogio la signal numero 1
El padre cogio la signal numero 2
El hijo cogio la signal numero 2
El padre cogio la signal numero 3
El hijo cogio la signal numero 3
El padre cogio la signal numero 4
El hijo cogio la signal numero 4
El padre cogio la signal numero 5
El hijo cogio la signal numero 5
El padre cogio la signal numero 6
El padre cogio la signal numero 6
El padre cogio la signal numero 7
El padre cogio la signal numero 7
El padre cogio la signal numero 8
El padre cogio la signal numero 8
El padre cogio la signal numero 9
El padre cogio la signal numero 9
El padre cogio la signal numero 10
El padre cogio la signal numero 10
El padre cogio la signal numero 11
El hijo cogio la signal numero 11
El padre cogio la signal numero 12
El hijo cogio la signal numero 12
El padre cogio la signal numero 13
El hijo cogio la signal numero 13
El padre cogio la signal numero 14
El hijo cogio la signal numero 14
El padre cogio la signal numero 15
El hijo cogio la signal numero 15
El padre cogio la signal numero 16
El hijo cogio la signal numero 16
El padre cogio la signal numero 17
El hijo cogio la signal numero 17
^C
```

En la variable antiguo se guarda la señal del padre, y de 5 a 10 se envía siempre esa señal, en vez de la del padre e hijo.

Modificacion:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <signal.h>
4 #include <sys/types.h>
5
6 int veces = 0; /* variable global */
7 void padre(int); /* funciones manejadoras */
8 void hijo(int);
9 int i = 0;
10 void (*antiguo)(int);
11
12 main()
13 {
14     pid_t creado, ppid;
15     signal(SIGUSR1, padre); /* pone SIGUSR1 */
16
17     switch(creado = fork())
18     {
19         case -1:
20             perror("fallo en el fork");
21             exit(1);
22         case 0: /* el hijo */
23             antiguo=signal(SIGUSR1, hijo);
24             ppid = getppid();
25             for(;;)
26             {
27                 if(i==5){
28                     antiguo=signal(SIGUSR1, antiguo);
29                 }
30                 if(i==10){
31                     signal(SIGUSR1, antiguo);
32                 }
33                 sleep(1); /* el otro hace el pause */
34                 /* envia signal al proceso padre */
35                 kill(ppid, SIGUSR1);
36                 i++;
37                 pause(); /* espero la respuesta */
38             }
39         default: /* el padre */
40             for(;;)
41             {
42                 pause(); /* espero a que me llamen */
43                 sleep(1); /* el hijo hace pause */
44                 kill(creado, SIGUSR1); /* aviso al hijo */
45             }
46     }
47
48     /* funcion de manejo de la signal para el padre */
49     void padre (int senal)
50     {
51         /* ambos procesos tienen una copia de veces */
52         printf("El padre cogio la signal numero %d\n", ++veces);
53         /* de nuevo para recibir la siguiente signal */
54         signal(senal, padre); /* pone SIGUSR1 para el padre */
55     }
56
57     /* funcion de manejo de la signal para el hijo */
58     void hijo (int senal)
59     {
60         /* ambos procesos tienen una copia de veces */
61         printf("El hijo cogio la signal numero %d\n", ++veces);
62         /* de nuevo para recibir la siguiente signal */
63         signal(senal, hijo); /* pone SIGUSR1 para el hijo */
64     }
65
66 }
```

22) Descárgate el programa agenda.c de la página, entiéndelo todos sus aspectos (ver comentarios), compílalo y prueba su funcionamiento.

```
diego@diego-VirtualBox:~/Desktop/Guion_2/Codigo/senales$ ./agenda 1 hola
Agenda: process-id 1907
diego@diego-VirtualBox:~/Desktop/Guion_2/Codigo/senales$ hola
```

23) Descárgate el programa bloqueo.c de la página, entiéndelo, compílalo y prueba su funcionamiento pulsando en control-C en las dos fases del programa dando uno o varios control-c ¿alguna diferencia?

[illegible]

No hay diferencia entre uno o varios ctrl-c.

24) Descárgate el programa bloqueo2.c de la página, entiéndelo, compílalo y prueba su funcionamiento pulsando en control-C en las dos fases del programa. Date cuenta que al poner en la funcion manejadora la opción por defecto el programa termina si se pulsa en la segunda fase.

```
diego@diego-VirtualBox:~/Desktop/Guion_2/Codigo/senales$ ./bloqueo2
^C^C^C^C^C^Ctenemos un control C pendiente
llego la señal
desbloqueada señal
^C
diego@diego-VirtualBox:~/Desktop/Guion_2/Codigo/senales$
```

25) Descárgate el programa sigac.c de la página, entiéndelo, compílalo y prueba su funcionamiento. Usa el man para darte cuenta de la complejidad de la llamada al sistema, dado que la acción tiene más campos y uno de ellos es otra estructura más amplia y compleja y que hay más opciones de las indicadas.

26) Descárgate el programa uno.c de la página, entiéndelo, compílalo y prueba su funcionamiento. Imagínate que no conoces el formato de lo que se ha escrito en la PIPE. Cambia la lectura de este programa para que sea genérica (lectura de 120 caracteres). Indica lo que sale por pantalla y búscale una explicación relacionada con las variables inicializadas y su ubicación. Cambia este programa suponiendo que no conocemos la cantidad de caracteres escritos. ¿qué ocurre?

Cambiamos el define a 120 y no varía nada, sin embargo, si pones un número menor al tamaño de los strings, solo se muestra ese número de palabras.

27) Descárgate el programa dos.c de la página, entiéndelo, compílalo y prueba su funcionamiento. Crea una copia y haz que la lectura sea realizada por dos procesos, con lo cual

tendrías un árbol con un abuelo, dos padres (uno lector y otro escritor) y un nieto. En el programa ya dado tenemos 3 procesos, abuelo padre y nieto, el padre escribe en pantalla el msg1 y el nieto el msg2 y el msg3. El programa termina pero no muestra el prompt

```

C:\Users\Victor\Dropbox\Guiones\Guion_2\Codigo\pipes\dos1.c - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
e19.c x Guion2.txt x dos1.c x
1 #include <stdio.h>
2 #include <stdlib.h>
3 #define tamano 40
4 main()
5 {
6     char *msg1 = "El primer mensaje";
7     char *msg2 = "El segundo mensaje";
8     char *msg3 = "El tercer mensaje";
9     char buffer[tamano];
10    int descriptor[2];
11    int i; creado, creado2;
12    /* se abre la pipe */
13    if (pipe(descriptor) < 0)
14    {
15        perror("Error en la llamada");
16        exit(1);
17    }
18
19    if ((creado=fork()) < 0)
20    {
21        perror("Error en el fork");
22        exit(2);
23    }
24    if (creado > 0)
25    {
26
27        if ((creado2=fork())<0){
28            perror("Error en el fork");
29            exit(3);
30        }
31        if (creado2 == 0){
32            close(descriptor[1]);
33            for (i=0;i<3;i++){
34                read(descriptor[0],buffer,tamano);
35                puts(buffer);
36            }
37        }
38    }
39    if (creado == 0)
40    {
41        close(descriptor[0]);
42        if ((creado=fork()) < 0)
43        {
44            perror("Error en el fork");
45            exit(4);
46        }
47        if (creado > 0)
48            write(descriptor[1],msg1, tamano);
49        if (creado==0)
50        {
51            write(descriptor[1],msg2, tamano);
52            write(descriptor[1],msg3, tamano);
53        }
54    }
55 }
56
57

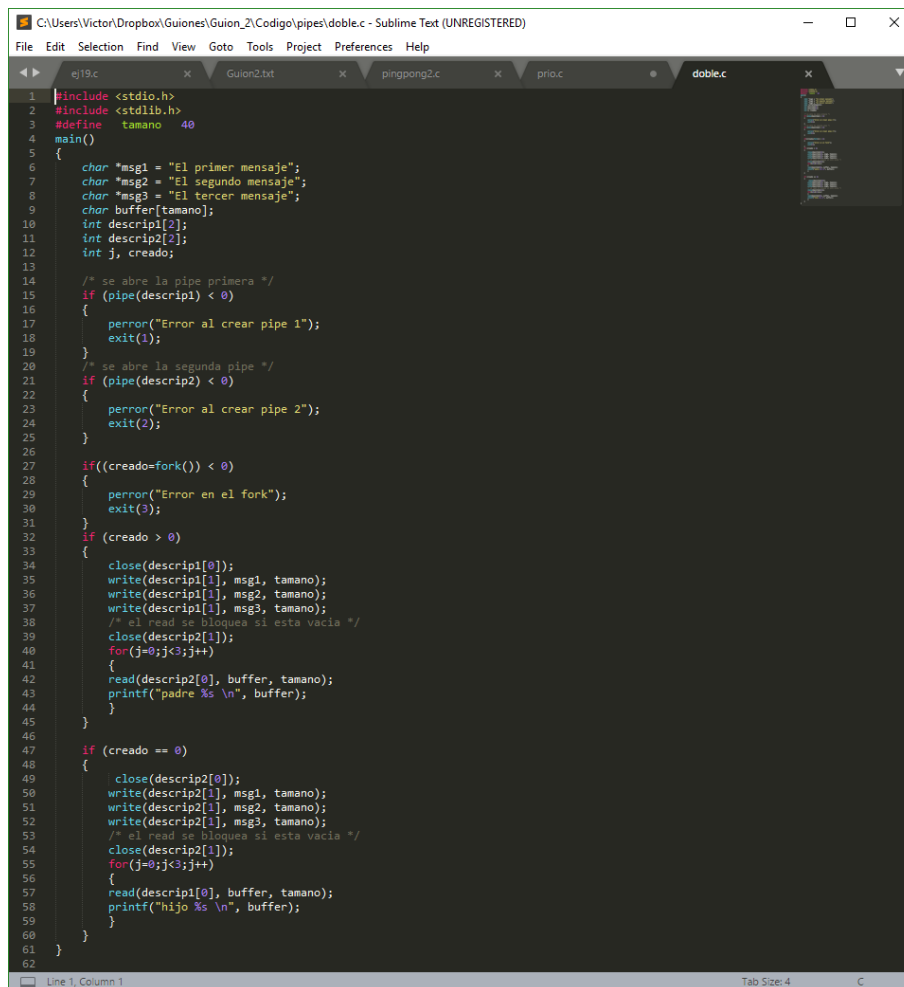
```

```

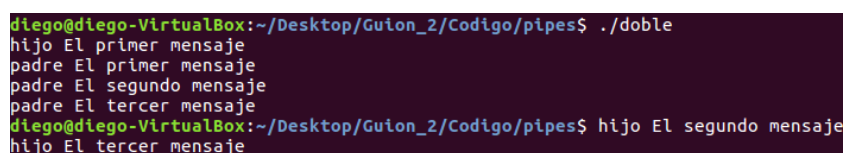
victor@victor-Lenovo-G50-80:/proc$ ps
PID TTY          TIME CMD
5536 pts/18      00:00:00 bash
6854 pts/18      00:00:00 ps
victor@victor-Lenovo-G50-80:/proc$ ls 5536/
attr          environ      mem          pagemap      stack
autogroup     exe          mountinfo   patch_state  stat
auxv          fd           mounts      personality  statm
cgroup        fdinfo       mountstats  proid_map    status
clear_refs    gid_map     net         root         syscall
cmdline       io          ns          sched        task
comm          limits      numa_maps   schedstat    timers
coredump_filter loginuid    oom_adj     sessionid    timerslack_ns
cpuset        map_files   oom_score   setgroups    uid_map
cwd           maps        oom_score_adj snaps         wchan
victor@victor-Lenovo-G50-80:/proc$

```

28) Descárgate el programa doble.c de la página, entiéndelo, compílalo y prueba su funcionamiento. El orden en la ejecución de los dos procesos dependerá de la planificación de los mismos. Cambia este programa para que los dos procesos empiecen escribiendo y después leyendo y comprueba el resultado.



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #define tamano 40
4 main()
5 {
6     char *msg1 = "El primer mensaje";
7     char *msg2 = "El segundo mensaje";
8     char *msg3 = "El tercer mensaje";
9     char buffer[tamano];
10    int descrip1[2];
11    int descrip2[2];
12    int j, creado;
13
14    /* se abre la pipe primera */
15    if (pipe(descrip1) < 0)
16    {
17        perror("Error al crear pipe 1");
18        exit(1);
19    }
20    /* se abre la segunda pipe */
21    if (pipe(descrip2) < 0)
22    {
23        perror("Error al crear pipe 2");
24        exit(2);
25    }
26
27    if ((creado=fork()) < 0)
28    {
29        perror("Error en el fork");
30        exit(3);
31    }
32    if (creado > 0)
33    {
34        close(descrip1[0]);
35        write(descrip1[1], msg1, tamano);
36        write(descrip1[1], msg2, tamano);
37        write(descrip1[1], msg3, tamano);
38        /* el read se bloquea si esta vacia */
39        close(descrip2[1]);
40        for(j=0;j<3;j++)
41        {
42            read(descrip2[0], buffer, tamano);
43            printf("padre %s \n", buffer);
44        }
45    }
46
47    if (creado == 0)
48    {
49        close(descrip2[0]);
50        write(descrip2[1], msg1, tamano);
51        write(descrip2[1], msg2, tamano);
52        write(descrip2[1], msg3, tamano);
53        /* el read se bloquea si esta vacia */
54        close(descrip2[1]);
55        for(j=0;j<3;j++)
56        {
57            read(descrip1[0], buffer, tamano);
58            printf("hijo %s \n", buffer);
59        }
60    }
61 }
62
```



```
diego@diego-VirtualBox:~/Desktop/Guion_2/Codigo/pipes$ ./doble
hijo El primer mensaje
padre El primer mensaje
padre El segundo mensaje
padre El tercer mensaje
diego@diego-VirtualBox:~/Desktop/Guion_2/Codigo/pipes$ hijo El segundo mensaje
hijo El tercer mensaje
```

29) Descárgate el programa tama.c de la página, entiéndelo, compílalo y prueba su funcionamiento. Indica cuál es el tamaño de tu sistema y si coincide con la constante PIPE\_BUF.

Se paro en 65536 y no se debería escribir mas de 4096.