



INSTITUTO POLITÉCNICO NACIONAL
UNIDAD PROFESIONAL INTERDISCIPLINARIA DE INGENIERÍA Y
CIENCIAS SOCIALES Y ADMINISTRATIVAS

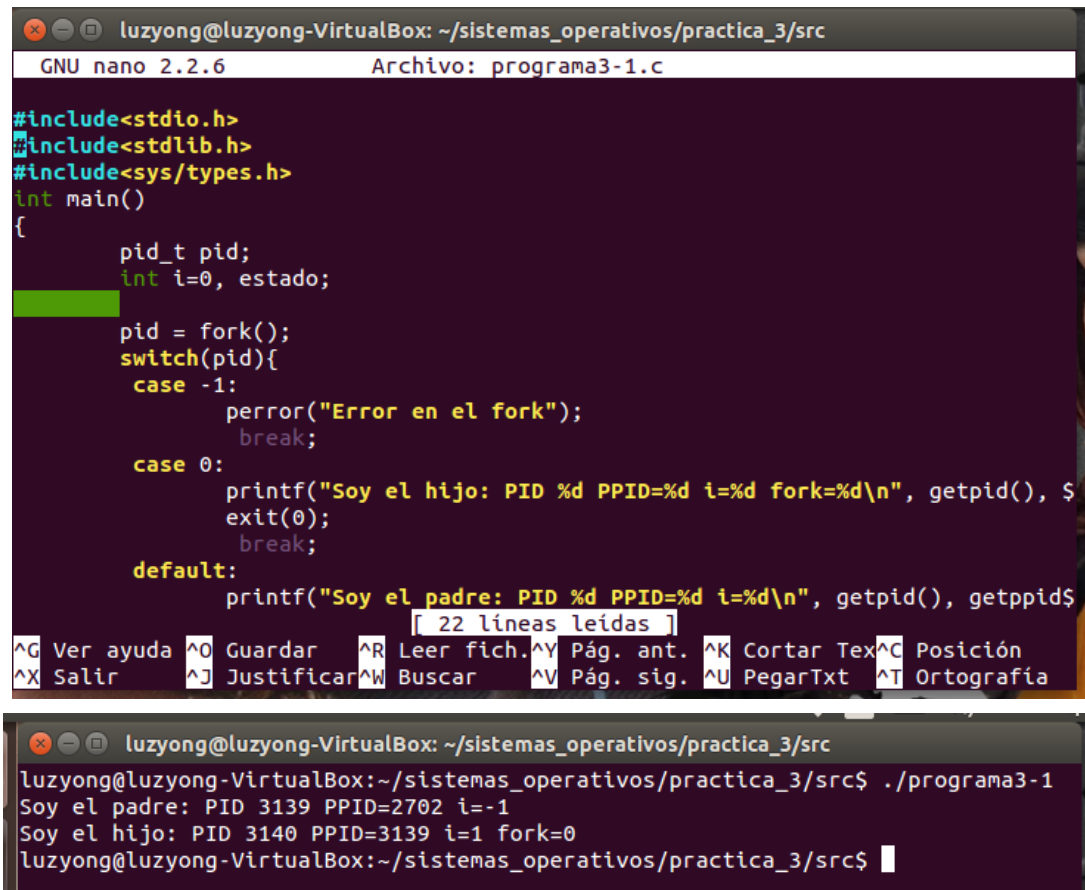


Práctica 3: Creación de procesos.

Yong Rodríguez Luz María

Programa 3-1

En este programa se crea un proceso nuevo con la llamada a `fork()` y se imprime dependiendo del valor que retorne `fork()` dentro de la variable `pid`.



```
luzyong@luzyong-VirtualBox: ~/sistemas_operativos/practica_3/src
GNU nano 2.2.6 Archivo: programa3-1.c

#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
int main()
{
    pid_t pid;
    int i=0, estado;

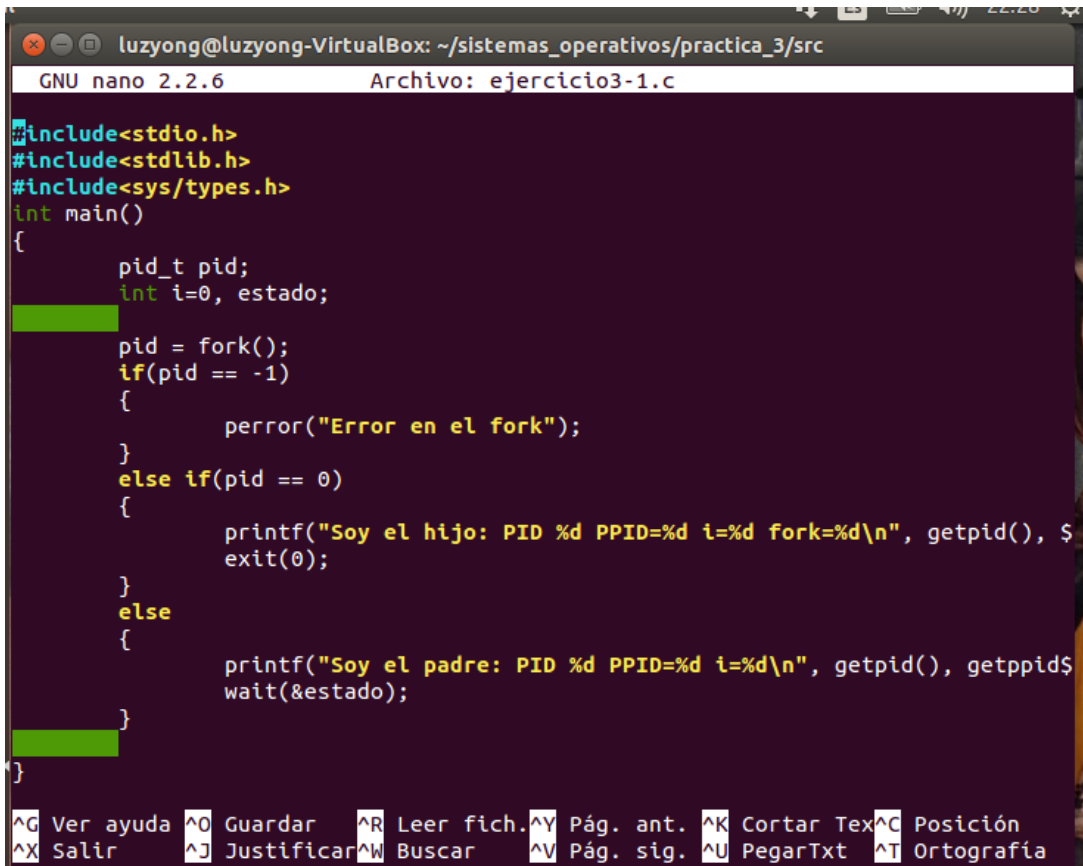
    pid = fork();
    switch(pid){
        case -1:
            perror("Error en el fork");
            break;
        case 0:
            printf("Soy el hijo: PID %d PPID=%d i=%d fork=%d\n", getpid(), $
            exit(0);
            break;
        default:
            printf("Soy el padre: PID %d PPID=%d i=%d\n", getpid(), getppid$
            [ 22 líneas leídas ]
^G Ver ayuda ^O Guardar ^R Leer fich.^Y Pág. ant. ^K Cortar Tex^C Posición
^X Salir ^J Justificar^W Buscar ^V Pág. sig. ^U PegarTxt ^T Ortografía

luzyong@luzyong-VirtualBox: ~/sistemas_operativos/practica_3/src
luzyong@luzyong-VirtualBox:~/sistemas_operativos/practica_3/src$ ./programa3-1
Soy el padre: PID 3139 PPID=2702 i=-1
Soy el hijo: PID 3140 PPID=3139 i=1 fork=0
luzyong@luzyong-VirtualBox:~/sistemas_operativos/practica_3/src$
```

- Pregunta 3-1. Si `fork` devolviese el mismo valor al proceso padre que al proceso hijo, ¿qué problemas le ocasionaría al programador?
Si el programador decidiera o tuviera que terminar un proceso, no sabría identificar cuál es el padre y cuál el hijo. Si terminamos con el proceso padre también lo hacemos con los hijos.
- Pregunta 3-2. Al ejecutar el código se imprimen tres identificadores de procesos. ¿A qué procesos corresponden dichos identificadores?
PID: el identificador del proceso, PPID: el identificador de su padre, `i`= el valor que devuelve `fork`.
- Pregunta 3-3. ¿Por qué no coincide dicha variable con el PID del proceso que imprime su valor?
Porque el PID es el identificador único de un proceso y puede cambiar de valor cada vez que se ejecute, la variable `i` siempre tendrá un valor de 0, -1 o cualquier valor diferente a esos números y son generados por la función `fork` al crear (o no) un proceso hijo.

Ejercicio 3-1.

Modifique el programa para que se utilicen funciones de decisión if-else en lugar de switch.

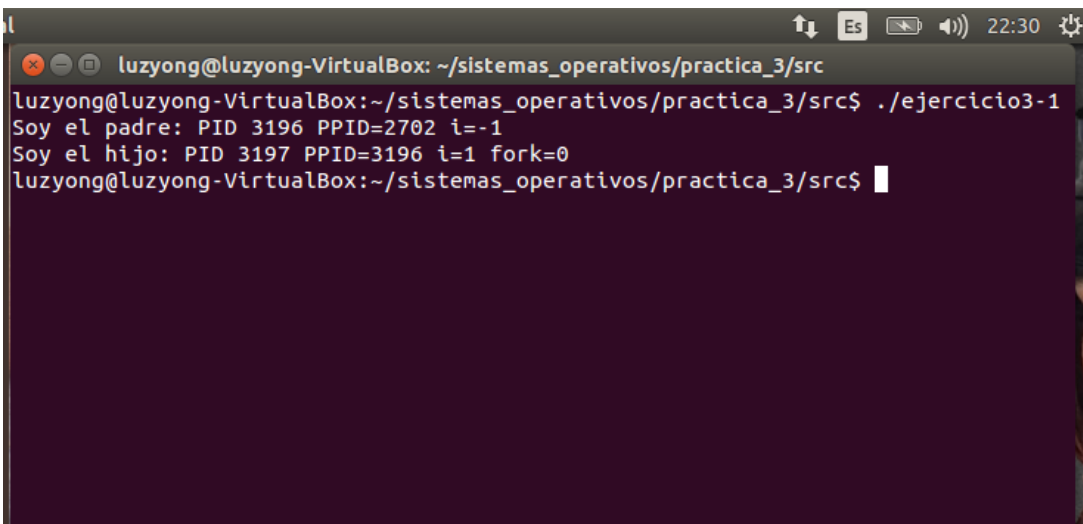


```
luzyong@luzyong-VirtualBox: ~/sistemas_operativos/practica_3/src
GNU nano 2.2.6 Archivo: ejercicio3-1.c

#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
int main()
{
    pid_t pid;
    int i=0, estado;

    pid = fork();
    if(pid == -1)
    {
        perror("Error en el fork");
    }
    else if(pid == 0)
    {
        printf("Soy el hijo: PID %d PPID=%d i=%d fork=%d\n", getpid(), $
        exit(0);
    }
    else
    {
        printf("Soy el padre: PID %d PPID=%d i=%d\n", getpid(), getppid$
        wait(&estado);
    }
}

^G Ver ayuda ^O Guardar ^R Leer fich. ^Y Pág. ant. ^K Cortar Tex ^C Posición
^X Salir ^J Justificar ^W Buscar ^V Pág. sig. ^U PegarTxt ^T Ortografía
```



```
luzyong@luzyong-VirtualBox: ~/sistemas_operativos/practica_3/src
luzyong@luzyong-VirtualBox:~/sistemas_operativos/practica_3/src$ ./ejercicio3-1
Soy el padre: PID 3196 PPID=2702 i=-1
Soy el hijo: PID 3197 PPID=3196 i=1 fork=0
luzyong@luzyong-VirtualBox:~/sistemas_operativos/practica_3/src$
```

Ejercicio 3-2.

Modifique el programa 3-1 de modo que el proceso hijo quede huérfano e imprima el PID de su padre adoptivo. En este código, en vez de utilizar la función wait, utilizo exit para que el proceso padre termine antes que su hijo.

```
luzyong@luzyong-VirtualBox: ~/sistemas_operativos/practica_3/src
GNU nano 2.2.6 Archivo: ejercicio3-2.c

#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
int main()
{
    pid_t pid;
    int i=0, estado;

    pid = fork();
    switch(pid){
        case -1:
            perror("Error en el fork");
            break;
        case 0:
            printf("Soy el hijo: PID %d PPID=%d i=%d fork=%d\n", getpid(), $
            exit(0);
            break;
        default:
            printf("Soy el padre: PID %d PPID=%d i=%d\n", getpid(), getppid$
            exit(0);
    }
}
```

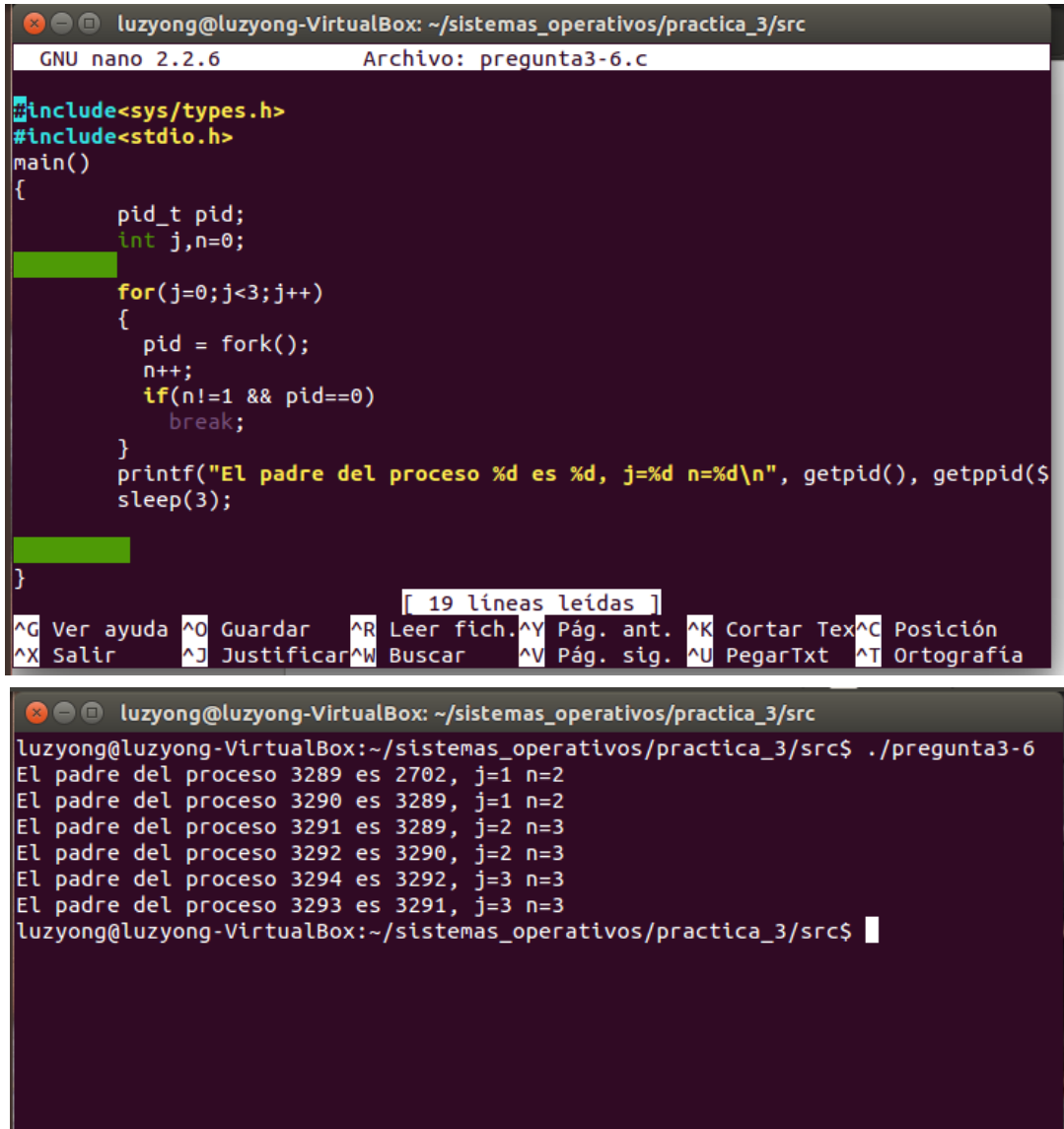
```
luzyong@luzyong-VirtualBox: ~/sistemas_operativos/practica_3/src
luzyong@luzyong-VirtualBox:~/sistemas_operativos/practica_3/src$ ./ejercicio3-2
Soy el padre: PID 3236 PPID=2702 i=-1
Soy el hijo: PID 3237 PPID=2194 i=1 fork=0
luzyong@luzyong-VirtualBox:~/sistemas_operativos/practica_3/src$
```

- Pregunta 3-4. ¿Cuántos procesos se ejecutan en total y cómo se encuentran enlazados entre ellos?
Se ejecutan 2 procesos en total y uno está enlazado a la terminal (el padre) y el otro al proceso init (el hijo).
- Pregunta 3-5. Si el ciclo for va desde 0 hasta 9, ¿por qué hay 11 impresiones?
Porque son 10 procesos creados mas el proceso padre.
- Pregunta 3-6. ¿Cuántos procesos se crearían si eliminamos las líneas consecutivas `if(pid != 0)` y `break;`? Los procesos que pueda aguantar la memoria. Break nos sirve para terminar un bucle y saltar a la siguiente línea de código. La condición nos dice que si el pid es diferente al del padre, entonces salga del bucle y vaya a la siguiente línea. Si quitamos esas condiciones, por cada proceso hijo va a crear otros n procesos hijos hasta

que todos los bucles de cada hijo que se convirtió en padre de otro proceso termine de iterar. Una fórmula general podría ser:

$$\#total_procesos = (\#procesos_deseados * \#procesos_padre) + \#procesos_padre.$$

En este código intento comprobar mi fórmula creando dos procesos padres.



The image consists of two screenshots of a terminal window. The top screenshot shows the source code of a C program named 'pregunta3-6.c' being edited in the nano text editor. The code includes headers for `<sys/types.h>` and `<stdio.h>`, and defines a `main()` function. Inside `main()`, it declares `pid_t pid;` and `int j, n=0;`. A `for` loop runs from `j=0` to `j<3`. Inside the loop, `pid = fork();` is called, `n` is incremented, and a `break;` statement is used to exit the loop after the first `fork()`. A `printf` statement prints the parent PID, the current `j` value, and the number of children `n`. A `sleep(3);` is also present. The bottom screenshot shows the execution of the program. The output displays the parent PID of each process and the number of children it has created, confirming the recursive nature of the process creation.

```
GNU nano 2.2.6 Archivo: pregunta3-6.c

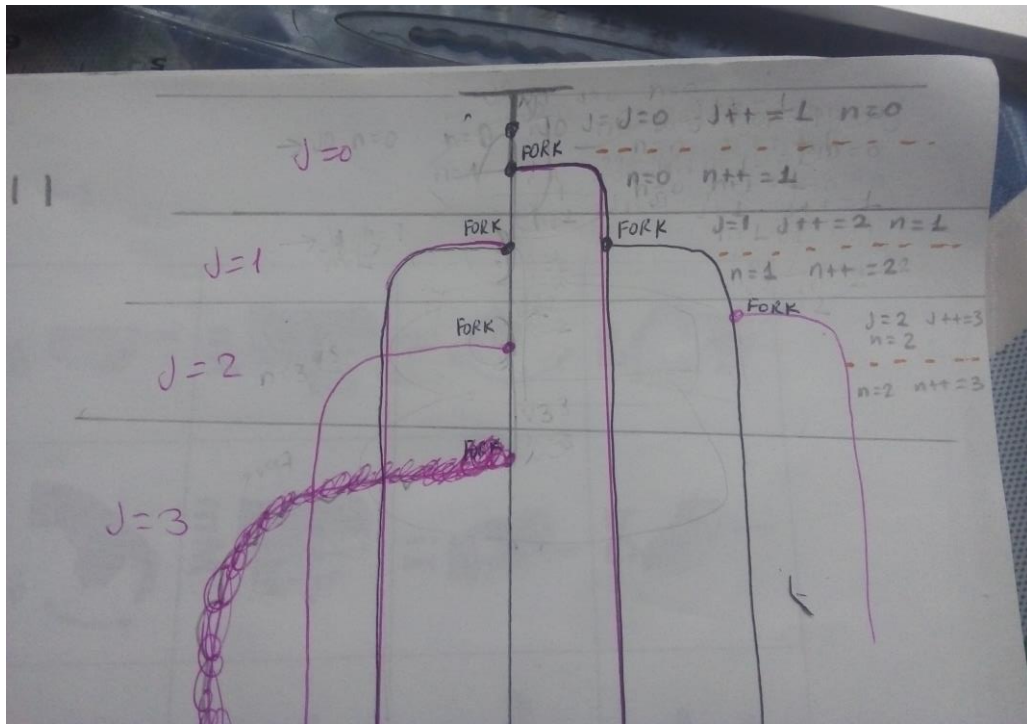
#include<sys/types.h>
#include<stdio.h>
main()
{
    pid_t pid;
    int j,n=0;

    for(j=0;j<3;j++)
    {
        pid = fork();
        n++;
        if(n!=1 && pid==0)
            break;
    }
    printf("El padre del proceso %d es %d, j=%d n=%d\n", getpid(), getppid($
sleep(3);

}

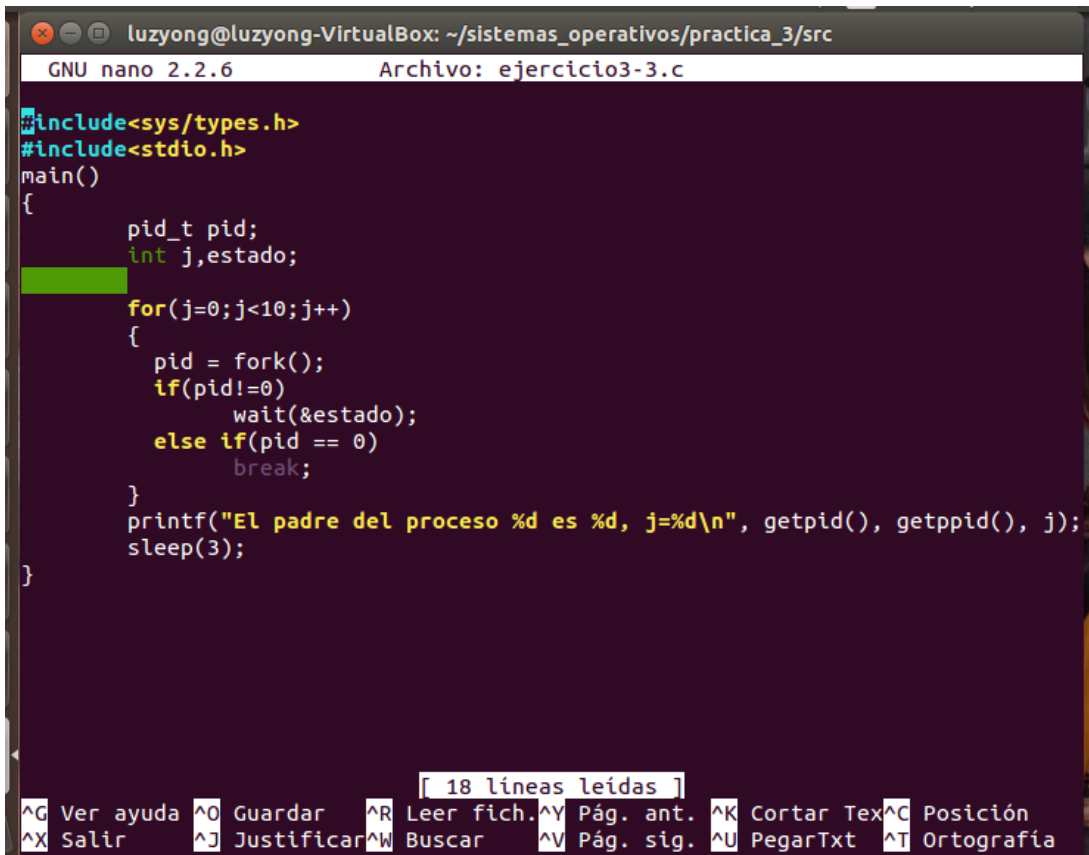
[ 19 líneas leídas ]
^G Ver ayuda ^O Guardar ^R Leer fich.^Y Pág. ant. ^K Cortar Tex ^C Posición
^X Salir ^J Justificar ^W Buscar ^V Pág. sig. ^U PegarTxt ^T Ortografía
```

```
luzyong@luzyong-VirtualBox: ~/sistemas_operativos/practica_3/src
luzyong@luzyong-VirtualBox:~/sistemas_operativos/practica_3/src$ ./pregunta3-6
El padre del proceso 3289 es 2702, j=1 n=2
El padre del proceso 3290 es 3289, j=1 n=2
El padre del proceso 3291 es 3289, j=2 n=3
El padre del proceso 3292 es 3290, j=2 n=3
El padre del proceso 3294 es 3292, j=3 n=3
El padre del proceso 3293 es 3291, j=3 n=3
luzyong@luzyong-VirtualBox:~/sistemas_operativos/practica_3/src$
```



Ejercicio 4-3.

Modifique el programa para que se generen 10 procesos y todos sean hijos del mismo padre, cada proceso debe imprimir su PID y el PID de su padre. En este código, evalúe el valor de fork. Utilizo wait para decirle al padre que espere a todos sus procesos hijos. Mientras el valor del pid sea el del padre, el bucle se va a repetir y va a crear procesos de un mismo padre. Si el pid es el del hijo, se sale e imprime el valor del que lo creó, así todos los procesos que se crearon son hijos de un mismo padre.



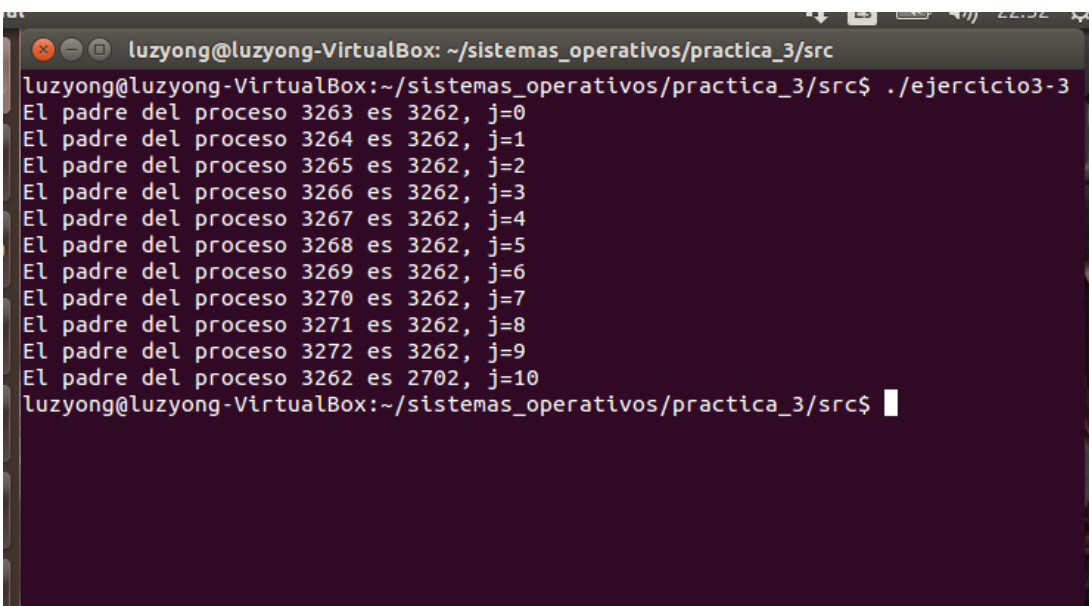
```
GNU nano 2.2.6 Archivo: ejercicio3-3.c

#include<sys/types.h>
#include<stdio.h>
main()
{
    pid_t pid;
    int j,estado;

    for(j=0;j<10;j++)
    {
        pid = fork();
        if(pid!=0)
            wait(&estado);
        else if(pid == 0)
            break;
    }
    printf("El padre del proceso %d es %d, j=%d\n", getpid(), getppid(), j);
    sleep(3);
}
```

[18 líneas leídas]

Ver ayuda Guardar Leer fich. Pág. ant. Cortar Tex Posición
Salir Justificar Buscar Pág. sig. PegarTxt Ortografía



```
luzyong@luzyong-VirtualBox: ~/sistemas_operativos/practica_3/src$ ./ejercicio3-3
El padre del proceso 3263 es 3262, j=0
El padre del proceso 3264 es 3262, j=1
El padre del proceso 3265 es 3262, j=2
El padre del proceso 3266 es 3262, j=3
El padre del proceso 3267 es 3262, j=4
El padre del proceso 3268 es 3262, j=5
El padre del proceso 3269 es 3262, j=6
El padre del proceso 3270 es 3262, j=7
El padre del proceso 3271 es 3262, j=8
El padre del proceso 3272 es 3262, j=9
El padre del proceso 3262 es 2702, j=10
luzyong@luzyong-VirtualBox:~/sistemas_operativos/practica_3/src$
```

Conclusiones.

En esta práctica aprendí a usar la función fork para crear procesos hijos. Aprendí lo que significan los valores que arroja y la diferencia que tiene con el PID. Aprendí que cuando el proceso padre muere, el proceso huérfano es adoptado por el proceso INIT.