INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE OCCIDENTE
Departamento de Electrónica, Sistemas e Informática.

Actividad 3: Procesos

Fundamentos de Sistemas Operativos
27/08/2024
Luis Antonio Pelayo Sierra
Luis Guillermo Rivera Stephens
Nombre del profesor: Jose Luis Elvira Valenzuela

Ejercicio 1

Descripción:

Desarrolla un programa donde un proceso padre crea un proceso hijo, al iniciar este proceso hijo, deberá matar o terminar a su padre.

Código fuente:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>

// Desarrolla un programa donde un proceso padre crea un proceso hijo, al iniciar
este proceso hijo, deberá matar o terminar a su padre.

int main()
{
    int p;

    p=fork();
    if(p==0)
    {
        kill(getppid(), SIGKILL);
        exit(0);
    }
    else {
        printf("Mi hijo esta a punto de ejecutarme\n");
    }
    wait(NULL);
    printf("Esta linea no deberia de imprimirse\n");
}
```

Ejecución:

```
fso@fsovm:~/Desktop/actividad-4$ ./ejer1
Mi hijo esta a punto de ejecutarme
Killed
fso@fsovm:~/Desktop/actividad-4$
```

Ejercicio 2

Descripción:

Desarrolla un programa donde un proceso padre crea un hijo, el padre tendrá una duración de 20 segundos y el hijo una duración de 1 segundo.

Ejecuta el proceso padre en segundo plano y antes de que el padre termine observa qué procesos hay.

$ ./ejer2 & ps

Código fuente:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>

// Desarrolla un programa donde un proceso padre crea un hijo, el padre tendrá
una duración de 20 segundos y el hijo una duración de 1 segundo.

int main()
{
    int p;
    p=fork();
    if (p==0) sleep(1);
    else sleep(20);
}
```

Ejecución:

```
fso@fsovm:~/Desktop/actividad-4$ ./ejer2 & ps u
[1] 2408
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
fso         2191  0.0  0.1  11640  5760 pts/0    Ss   12:32   0:00 /bin/bash
fso         2408  0.0  0.0   2548  1152 pts/0    S    12:45   0:00 ./ejer2
fso         2409  600  0.1  13612  4480 pts/0    R+   12:45   0:00 ps u
fso         2410  0.0  0.0   2548   128 pts/0    S    12:45   0:00 ./ejer2
fso@fsovm:~/Desktop/actividad-4$ ps u
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
fso         2191  0.0  0.1  11640  5760 pts/0    Ss   12:32   0:00 /bin/bash
fso         2408  0.0  0.0   2548  1152 pts/0    S    12:45   0:00 ./ejer2
fso         2410  0.0  0.0      0     0 pts/0    Z    12:45   0:00 [ejer2] <defunct>
fso         2412  600  0.1  13612  4480 pts/0    R+   12:45   0:00 ps u
fso@fsovm:~/Desktop/actividad-4$ ps u
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
fso         2191  0.0  0.1  11640  5760 pts/0    Ss   12:32   0:00 /bin/bash
fso         2414  700  0.1  13612  4480 pts/0    R+   12:46   0:00 ps u
[1]+  Done                    ./ejer2
fso@fsovm:~/Desktop/actividad-4$
```

Ejercicio 3

Descripción:

Desarrolla un programa donde un proceso padre crea un hijo, el padre tendrá una duración de 1 segundo y el hijo una duración de 20 segundos.

Ejecuta el proceso padre en segundo plano y antes de que el padre termine observa qué procesos hay.

$ ./ejer3 & ps

Código fuente:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>

// Desarrolla un programa donde un proceso padre crea un hijo, el padre tendrá una duración de 1 segundo y el hijo una duración de 20 segundos.

int main()
{
    int p;
    p=fork();
    if (p==0) sleep(20);
    else sleep(1);
}
```

Ejecución:

```
fso@fsovm:~/Desktop/actividad-4$ ./ejer3 & ps u
[1] 2429
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
fso         2191  0.0  0.1  11640  5760 pts/0    Ss   12:32   0:00 /bin/bash
fso         2429  0.0  0.0   2548  1152 pts/0    S    12:50   0:00 ./ejer3
fso         2430  600  0.1  13612  4480 pts/0    R+   12:50   0:00 ps u
fso         2431  0.0  0.0   2548   128 pts/0    S    12:50   0:00 ./ejer3
fso@fsovm:~/Desktop/actividad-4$ ps u
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
fso         2191  0.0  0.1  11640  5760 pts/0    Ss   12:32   0:00 /bin/bash
fso         2431  0.0  0.0   2548   128 pts/0    S    12:50   0:00 ./ejer3
fso         2432  350  0.1  13612  4480 pts/0    R+   12:50   0:00 ps u
[1]+  Done                    ./ejer3
fso@fsovm:~/Desktop/actividad-4$ ps u
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
fso         2191  0.0  0.1  11640  5760 pts/0    Ss   12:32   0:00 /bin/bash
fso         2433  500  0.1  13612  4480 pts/0    R+   12:51   0:00 ps u
fso@fsovm:~/Desktop/actividad-4$
```

Ejercicio 4

Descripción:

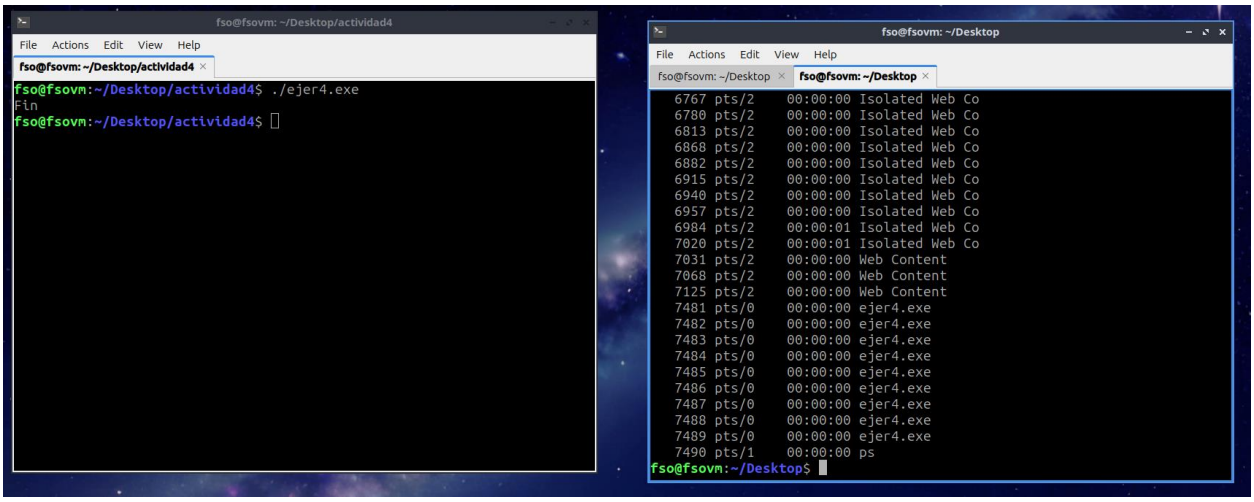
Desarrolla un programa que al ejecutarse cree dos procesos hijos y cada uno de los hijos creará 3 nuevos procesos hijos. De esta manera habrá un proceso padre, dos procesos hijos y seis procesos nieto. Los hijos y los nietos deberán tener al final la instrucción sleep(20), pero el padre después de ejecutar la instrucción sleep(5) deberá terminar a todos los hijos y nietos.

Código fuente:

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

int main()
{
   int p;
   p=fork();
   int p2 = p;    // Respaldar pid del hijo 1
   if (p!=0) p=fork(); // Si soy el padre, tengo mi segundo hijo
   if (p==0){  // Si soy uno de los dos hijos...
      p=fork(); // Tendre mi primer hijo
      if (p!=0) p=fork(); // Mi segundo hijo
      if (p!=0) p=fork(); // Mi tercer hijo
      sleep(20); // y todos dormiremos 20 segundos
   }
   else { // Mientras tanto, el padre
      sleep(5); // dormira 5 segundos
      kill(-p, SIGKILL); // Assesina todos los procesos del mismo grupo
   }
   printf("Fin\n"); // Solo se deberia imprimir 1 vez
}
```

Ejecución:



Preguntas:

- Si ejecuto un proceso que cree otros procesos, ¿Cómo puedo ver que procesos se están ejecutando?
  Con el comando en terminal ps

- De los ejercicios 2 y 3, ¿En qué casos hubo procesos zombies?
  En el ejercicio 2, pues los procesos hijos no terminan hasta que el papá muere

- ¿Qué es un proceso zombie?, ¿conviene evitarlos?, ¿por qué?

  Un proceso zombie es un proceso finalizado que ya no se ejecuta pero que sigue reconociéndose en la tabla de procesos (en otras palabras, tiene un número PID). Ya no se asigna espacio del sistema a dicho proceso. Los procesos zombie han sido cerrados o han salido y siguen existiendo en la tabla de procesos hasta que muere el proceso padre.

  Si se generan demasiados procesos zombies, la tabla de procesos estará llena. Es decir, el sistema no podrá generar ningún proceso nuevo, entonces el sistema se detendrá. Por tanto, debemos evitar la creación de procesos zombies.

  https://www.ibm.com/docs/es/aix/7.1?topic=processes-

  https://www-geeksforgeeks-org.translate.goog/zombie-processes-prevention/?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=rq#:~:text=If%20too%20many%20zombie%20processes,will%20come%20to%20a%20standstill.

- ¿Qué aprendiste?

Profundizamos en como funcionan los procesos y como se “matan”, también vimos un nuevo tipo de proceso o estatus llamado zombie. Lo más difícil fue intentar el retador, puesto que tienes que manejar procesos nietos, aunque encontramos la solución con el uso de los group process usando -pid en kill para eliminar los procesos hijos y nietos “If *pid* is negative, but not -1, *sig* shall be sent to all processes (excluding an unspecified set of system processes) whose process group ID is equal to the absolute value of *pid*, and for which the process has permission to send a signal.”
https://pubs.opengroup.org/onlinepubs/009695399/functions/kill.html