Guía práctica de procesos

En sistemas Unix/Linux, la principal herramienta para el control de procesos es el *shell*, que permite gestionar un proceso en ejecución: pasarlo a segundo plano, suspenderlo, activarlo, desasociarlo de la terminal, detenerlo, etc.

1. Ejecute el comando **`yes`** en la terminal y luego detenga el comando con **`Ctrl+Z`**. Identifique el número identificador de trabajo asignado.

2. Mediante el comando **`fg`** (*foreground*) podrá recuperar y volver a ejecutar el proceso asignado. Ejecute **`fg`** acompañado del número de proceso **`fg %1`**.

3. Mediante el comando **`bg`** (*background*) podrá envíar y volver a ejecutar el proceso asignado a segundo plano. Ejecute **`bg`** acompañado del número de proceso **`bg %1`**.

4. Lance el comando **`yes`** en segundo plano, ejecutando **`yes > /dev/null &`**.

5. Utilice el comando **`jobs`** para mostrar el listado de procesos que están en el sistema junto con su estado e identificador.

6. Utilice el comando **`kill`** para detener los procesos listados en el punto 5.

7. Utilice el comando **`ps`** para listar los procesos en ejecución. La primera columna es el **`PID`** del proceso el cual nos permitirá identificarlos.

8. Ingrese el comando **`ps f`**, ¿nota la diferencia? La opción **`f`** como la **`l`** tienen una columna **`STAT`** que indica el estado del proceso. Los posibles estados que puede indicar son:
    - **`D (Disk sleep)`**: el proceso está durmiendo; ininterrumpible (generalmente a la espera por E/S).
    - **`R (Running)`**: el proceso se está ejecutando en la CPU (o en cola de listos).
    - **`S (Sleeping)`**: el proceso está durmiendo, no se está ejecutando en la CPU en este momento.
    - **`T (Stopped)`**: el proceso está detenido, por una señal de usuario o por otra causa, pero se puede reiniciar más tarde.
    - **`W (Paging)`**: el proceso está paginado, a la esperando que se le asignen recursos de memoria.
    - **`X (Dead)`**: el proceso está muerto, ha sido eliminado o ha finalizado de forma anormal.
    - **`N (New)`**: el proceso ha sido creado recientemente y aún no se ha ejecutado.
    - **`Z (Zombie)`**: defunct ("zombie")

9. El comando **top** permite ver los procesos del sistema y estadísticas de los mismos. Ejecute el comando **man top**.

10. Para crear nuevos procesos, los sistemas UNIX y Linux disponen únicamente de una llamada al sistema, **fork**, sin ningún tipo de parámetros: `int fork()`;

    Al llamar a esta función se crea un nuevo proceso (proceso hijo), idéntico en código y datos al proceso que ha realizado la llamada (proceso padre). Inicialmente el proceso hijo se diferencia de su padre únicamente por su **PID** (Identificador de Proceso) y por su **PPID** (*Parent PID*). Se puede distinguir si se es el proceso padre o el hijo por medio del valor de retorno de **fork**. Esta función devuelve un cero al proceso hijo, y el identificador de proceso (**PID**) del hijo al proceso padre. Cómo se garantiza que el **PID** siempre es no nulo, basta aplicar un **if** para determinar quién es el padre y quién el hijo para así ejecutar distinto código.

    Compile el siguiente programa y ejecútelo en segundo plano en una terminal.

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

int main()
{
    pid_t pid;
    int i;

    pid = fork();
    // La función fork(); retorna 0 a los procesos hijos.
    if (pid == 0)
    {   // Proceso hijo...
        for (i = 0; i < 10; i++)
        {
            printf("Hijo: %d\n", i);
            sleep(1);
        }
        return 0;
    }
    else if (pid > 0)
    {   // Proceso padre...
        for (i = 0; i < 10; i++)
        {
            printf("Padre: %d\n", i);
            sleep(1);
        }
        return 0;
    }
    else
    {
        printf("No se ha podido bifurcar");
    }
    return 0;
}
```

Utilizando el comando **ps**, muestre cuales son los procesos que están ejecutándose. Realice lo mismo pero ahora duplicando la línea donde se invoca a la llamada **fork()**. ¿Cuáles son sus conclusiones?.

NOTA: *En la línea 10 se produce la bifurcación de procesos, se tendrán dos procesos de los cuales uno es hijo de otro. Luego, se muestra durante 10 veces el mensaje "hijo" y "padre" en bucles independientes. Tener en claro que no se producen nuevos procesos luego de la bifurcación de la línea 10 debido a que la misma está fuera de los bucles.*

11. Indique cuántas veces se muestra la palabra FICH al ejecutar el programa mostrado a continuación. Describa su funcionamiento.

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/types.h>
4
5  int main()
6  {
7      fork();
8      fork();
9      fork();
10     printf("FICH\n");
11     return 0;
12 }
13
```

Generalice el código para **n-forks**. Analice para **n=1**, luego para **n=2**, etc., determine la serie y deduzca la expresión general en función de **n**.

12. Pruebe el *Programa1* y compárelo con *Programa2*, ¿a qué conclusiones llega?.

Programa1:

```
5  int main()
6  {
7      unsigned int i, j, a, v[50000];
8      for (i = 0; i < 50000; i++)
9          v[i] = i * i;
10
11     printf("\nInicio\n");
12     for (i = 0; i < 50000 - 1; i++)
13         for (j = i + 1; j < 50000; j++)
14             if (v[i] < v[j])
15             {
16                 a = v[i];
17                 v[i] = v[j];
18                 v[j] = a;
19             }
20     printf("\nOrdenamiento terminado!\n\n");
21     execlp("/bin/ls", "ls", NULL);
22     return 0;
23 }
```

Programa2:

```c
int main()
{
    pid_t pid;
    unsigned int i, j, a, v[50000];

    for (i = 0; i < 50000; i++)
        v[i] = i * i;

    pid = fork();
    if (pid == 0)
    {
        printf("\nInicio\n");
        for (i = 0; i < 50000 - 1; i++)
            for (j = i + 1; j < 50000; j++)
                if (v[i] < v[j])
                {
                    a = v[i];
                    v[i] = v[j];
                    v[j] = a;
                }
        printf("\nOrdenamiento terminado!\n\n");
        return 0;
    }
    else
    {
        execlp("/bin/ls", "ls", NULL);
        return 0;
    }
}
```

13. Pruebe y analice el siguiente código y arme el árbol de procesos generado.

```c
int main(void)
{
    int i;
    int padre = 1;
    for (i = 0; i < 3; i++)
    {
        if (padre == 1)
        {
            if (fork() == 0) // Proceso hijo
            {
                printf("Proceso hijo  PID:%d con padre PPID:%d\n",
                       getpid(), getppid());
                padre = 0;
            }
            else            // Proceso padre
            {
                printf("Proceso padre PID:%d\n", getpid());
                padre = 1;
            }
        }
        sleep(1);
    }
    return 0;
}
```

14. Puede ver el estado de un proceso ejecutando el programa en una terminal (por ejemplo `./Ejercicio13.bin`) y en otra escribir:

```
watch -n 0.5 'ps -A -e -o ppid,pid,stat,cmd,tty | grep -i Ejercicio13'
```

15. Considere el siguiente programa y cree el árbol correspondiente:

```
1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5
6  using namespace std;
7
8  int main()
9  {
10     int i;
11     for (i = 0; i < 3; i++)
12     {
13         if (fork() != 0)
14             printf("Proceso %d hijo de %d\n", getpid(), getppid());
15         wait(NULL);
16     }
17     return 0;
18 }
```

16. Considere el siguiente programa y cree el árbol correspondiente:

```
1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <unistd.h>
4
5  int main()
6  {
7      int i;
8      printf("Inicio\n");
9      for (i = 0; i < 4; i++)
10         if (fork() == 0)
11             break;
12     sleep(30);
13     return 0;
14 }
```

Realiza lo mismo pero cambiando la condición del **`if`** por **`(fork() != 0))`**

17. Considere el siguiente programa y determine su estructura y su salida:

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

int main()
{
    int num, pid;
    for (num = 1; num <= 3; num++)
    {
        pid = fork();
        if ((num == 3) && (pid == 0))
        {
            printf("\n");
            execlp("ls", "ls", "-l", NULL);
        }
        wait(NULL);
    }
    return 0;
}
```

Realiza lo mismo pero quitando el **if** y analice lo que sucede.