

PROCESOS EN LINUX

Procesos en Linux (Introducción)

Linux:

- Mantiene constancia de cada proceso del sistema en una estructura de datos denominada `task_struct`
- La declaración de la estructura de datos se encuentra en el archivo de cabecera `~/include/linux/sched.h`

Entre los atributos de un proceso, se encuentran:

- Identificador (PID): Número único que el sistema asigna a cada proceso de tal forma que aunque los PID se puedan repetir en el sistema, nunca pueden hacerlo en el mismo momento.
- Identificador del proceso padre (PPID).
- Identidades del usuario y grupos.
- Prioridad del proceso con respecto a otros.
- Recursos consumidos por el proceso.
- Archivos abiertos
- Estado del proceso: Especifica la actividad que esta realizando el proceso.

Procesos especiales:

Swapper(scheduler process):

- Proceso 0
- Parte del kernel

Init:

- Proceso 1
- Invocado por el kernel
- Lee `/etc/rc*` donde están los archivos de inicialización
- Nunca muere

Ingresa al sistema como *usuario no privilegiado*. Cree un directorio donde almacenar los programas de este práctico dentro de su directorio HOME. Luego sitúese en el directorio recién creado.

Escriba el siguiente programa y nómbrelo “Prog_0501.pp” y compílelo.

```
program Prog_0501;  
  
Uses BaseUnix;  
  
begin  
  
    Writeln(' Identificador de proceso: ', fpgetpid);  
    Writeln(' Identificador del proceso padre: ', fpgetppid);  
  
end.
```

(1) Ejecute el programa varias veces y compare los resultados. Qué hay diferente entre cada ejecución? Qué hay igual? Por qué ocurre esto?

Escriba el siguiente programa y nómbrelo “Prog_0502.pp” y compílelo¹.

```
program Prog_0502;  
  
Uses BaseUnix;  
  
begin  
  
    Writeln('Identificador de usuario: ', fpgetuid);  
    Writeln('Identificador efectivo de usuario: ', fpgeteuid);  
    Writeln('Identificador de grupo: ', fpgetgid);  
    Writeln('Identificador efectivo de grupo: ', fpgetegid);  
  
end
```

Este programa hace uso de las llamadas POSIX para obtener el identificador actual y efectivo de usuario y de grupo. Ejecute el programa y observe los resultados.

Cree dos copias del programa ejecutable con los nombres “Prog_0502_suid” y “Prog_0502_sgid”.

Ejecute los comandos `chmod u+s Prog_0502_suid` y `chmod g+s Prog_0502_sgid`.

Ejecute el comando `sudo su` y escriba la contraseña de usuario. (Ahora usted es el usuario root).

(2) Ejecute los tres programas y compare sus salidas. A qué se deben las diferencias? (Vea la página man del comando `chmod` para más información).

Ejecute el comando `exit` para volver a trabajar como su usuario.

Escriba el siguiente programa y nómbrelo “Prog_0503.pp” y compílelo.

```
program Prog_0503;  
  
Uses BaseUnix;  
  
begin  
  
    Writeln('El directorio actual es: ', fpgetenv('PWD'));  
    Writeln('El Path es: ', fpgetenv('PATH'));  
  
end.
```

Este programa hace uso de las llamadas POSIX para obtener variables de entorno del proceso actual. Ejecute el programa y observe los resultados.

(3) Cambie al directorio padre del actual y vuelva a ejecutar el programa, compare la salida actual con la ejecución anterior. A qué se debe la diferencia en la salida?

¹A partir de aquí NO se puede compilar a través del propio IDE (con alt+F9) deberá hacerse desde la consola: `fpc <fuente a compilar>`

Creación de procesos

Para crear un proceso desde otro se utiliza la llamada al sistema `fork`:

- Permite crear procesos desde otro.
- Esta función o llamada al sistema permite crear procesos hijos que se diferencian del padre en su identificador de proceso.
- El sistema duplica en memoria el contenido del espacio virtual asignado al proceso padre en otro espacio para el proceso hijo, de tal manera, que ambos espacios virtuales sean independientes, no comparten la memoria.

Esta función tiene tres posibles valores de retorno:

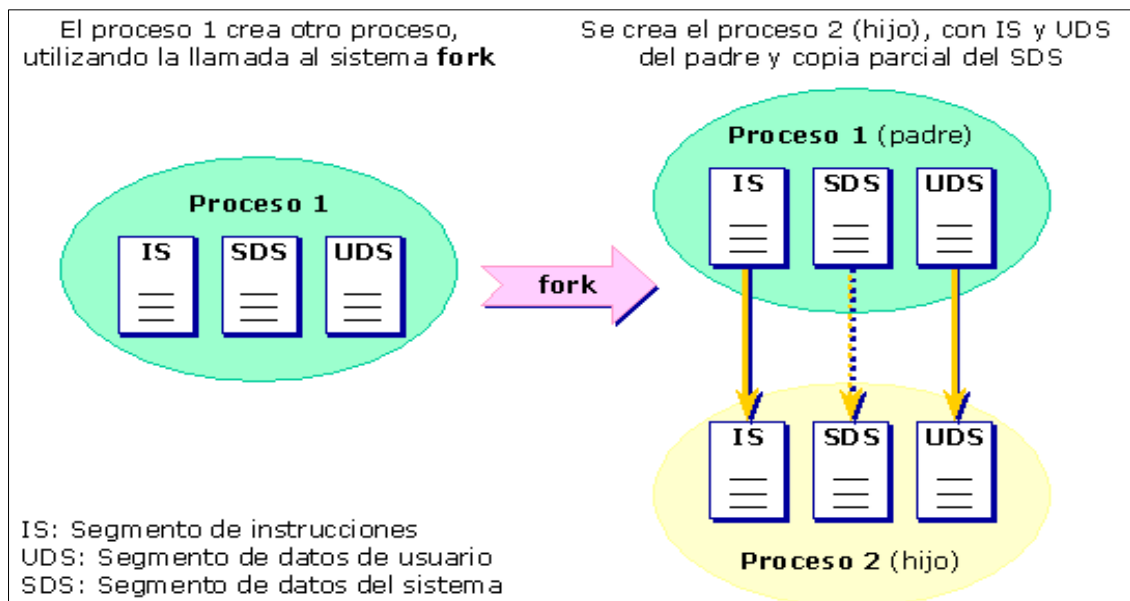
- Igual 0: Este código lo recibe el proceso hijo.
- Mayor que 0: Este código lo recibe el proceso que ha ejecutado la llamada al sistema, es decir el proceso padre. Y su valor es el número de proceso que identifica al proceso hijo obtenido como resultado de la llamada.
- Menor que 0: La llamada al sistema no ha podido crear un proceso.

La función `fork()` devuelve el process ID del nuevo proceso (proceso hijo) al padre.

- Porque el proceso padre puede tener mas de un hijo.
- El hijo puede obtener su process ID por medio de la función `fpgetpid()`

Devuelve 0 al proceso hijo

- Porque un proceso hijo solo puede tener un padre
- El proceso hijo puede obtener el process ID del padre por medio de la función `fpgetppid()`



Creación de procesos con `fork()`.

Escriba el siguiente programa y nómbrelo “Prog_0505.pp” y compílelo.

```
program Prog_0505;

Uses BaseUnix;

var
    pid: Longint;

begin
    pid := fpFork;

    case pid of
        -1 : Writeln('Error en el fork!!!');
        0 : begin
                Writeln(' Soy el hijo:');
                Writeln(' Id del proceso: ', fpgetpid);
                Writeln(' Id de mi proceso padre: ', fpgetppid);
            end;
        else begin
                Writeln(' Soy el padre:');
                Writeln(' Id del proceso: ', fpgetpid);
                Writeln(' Id de mi proceso padre: ', fpgetppid);
            end;
    end;
end;
```

Este programa hace uso de la llamada POSIX para realizar un fork del proceso actual.

(5) Ejecute el programa varias veces (unas 20) y compare los resultados. Qué hay diferente entre cada ejecución? Qué hay igual? Por qué ocurre esto?

Escriba el siguiente programa y nómbrelo “Prog_0506.pp” y compílelo.

```
program Prog_0506;

Uses BaseUnix, Crt;

var
    pid: Longint;

begin
    pid := fpFork;

    case pid of
        -1 : Writeln('Error en el fork!!!');
        0 : begin
                Writeln(' Soy el hijo, y mi PID es: ', fpgetpid, '. Y el PID de mi padre es: ',
                    fpgetppid);
                Delay(50000);
            end;
        else begin
                Writeln(' Soy el padre, y mi PID es: ', fpgetpid);
            end;
    end;
end.
```

Este programa hace uso de la llamada POSIX para realizar un fork del proceso actual.

(6) Ejecute el programa e inmediatamente ejecute el comando " ./Prog_0506; ps -f" y observe los resultados. Según la salida del ps quién es el padre del proceso hijo? Quién debería serlo? Por qué ocurre esto?

Escriba el siguiente programa y nómbrelo "Prog_0507.pp" y compílelo.

```
program Prog_0507;

Uses BaseUnix;

var
  pid: Longint;
  a: Integer;

begin
  a:=5;
  pid := fpFork;

  case pid of
    -1 : Writeln('Error en el fork!!!');
    0 : begin {Proceso hijo}
          a:=a+1;
          Writeln(' (Hijo) El valor de a es: ', a);
        end;
    else begin {Proceso padre}
          a:=a*2;
          Writeln(' (Padre) El valor de a es: ', a);
        end;
  end;
end.
```

(7) Ejecute el programa y observe los resultados. Por qué la variable "a" muestra diferentes valores en el padre y en el hijo?

Escriba el siguiente programa y nómbrelo "Prog_0508.pp" y compílelo.

```
program Prog_0508;

Uses BaseUnix;

var
  pid: Longint;
  i, n: Integer;

begin
  n := 5;

  for i:=1 to n do
    begin
      pid := fpFork;
      if pid <> 0 then break;
    end;
    Writeln('El padre del proceso ', fpgetpid, ' es: ', fpgetppid);
  end;
end.
```

Ejecute el programa varias veces y observe los resultados.

(8) Modifique el programa anterior para que el proceso original sea el padre de todos los procesos creados por el fork y guarde esta versión modificada como "Prog_0509.pp".

Qué modificaciones requirió el programa para comportarse de esta manera? Por qué?

Escriba el siguiente programa y nómbrelo "Prog_0510.pp" y compílelo.

```
Program Prog_0510;
Uses BaseUnix, unix;
var
    pid: Tpid;
    options: longint;
    status: longint;
begin
    pid := fpFork;
    options:=0;
    status:=0;
    case pid of
        -1 : Writeln('Error en el fork!!!');
        0 : begin
                {Proceso hijo }
                Writeln('Proceso hijo:');
                fpExecl('/bin/ls', ['-l']);
            end;
        else begin
                {Proceso padre }
                fpWaitpid(pid, status, options);
                Writeln('Proceso padre. ');
            end;
    end;
end;
```

Este programa hace uso de las llamadas POSIX para crear procesos a partir de un ejecutable y la llamada `Waitpid` para que el proceso padre espere a su hijo antes de salir. Ejecute el programa y observe los resultados.

(9) Cambie al directorio padre del actual y vuelva a ejecutar el programa, compare la salida actual con la ejecución anterior. Nota alguna diferencia? Por qué?

Elabore un informe conteniendo la respuesta a cada punto escrito entre paréntesis.

Escriba el siguiente programa y nómbrelo “Prog_0512.pp” y compílelo.

```
program Prog_0512;

uses BaseUnix;

var
    pid: Longint;

begin
    pid := Fork;

    case pid of
        -1 : Writeln('Error en el fork!!!');
        0  : begin {Proceso hijo }
                Writeln('PID antes del EXEC: ', fpgetpid);
                Writeln('PPID antes del EXEC: ', fpgetppid);
                Writeln('Ejecutando EXEC...');
                Exec('Prog_0501');
                Writeln('Terminado el EXEC...');
            end;
        else begin {Proceso padre }
                WaitProcess(pid);
            end;
    end;
end;
```

(11) Ejecute el programa y observe los resultados. Por qué se mantienen los mismos valores del PID y del PPID en el proceso creado por EXEC a partir del ejecutable Prog_0501? Por qué no apareció el mensaje “Terminado el EXEC...”?

Confeccione un informe **original, conciso y completo** donde se de respuesta a las preguntas y consignas precedidas por un número encerrado entre parentesis. Este informe deberá ser confeccionado y entregado individualmente por los miembros de cada grupo. La fecha límite de entrega es el **12/09/2015** a las **0:00** horas. La longitud máxima del informe es de tres paginas (sin contar las líneas correspondientes a código fuente).

Escriba el siguiente programa y nómbrelo “Prog_0513.pp” y compílelo.

```
program Prog_0513;

Uses BaseUnix, unix;

Const
  Arg0: PChar = '/bin/ls';
  Arg1: PChar = '-l';

var
  pid, options: Longint;
  status: longint;
  PP: PPchar;

begin
  GetMem(PP, 3*SizeOf(Pchar));

  PP[0] := Arg0;
  PP[1] := Arg1;
  PP[2] := Nil;

  pid := fpFork;
  status := 0;
  options := 0;

  case pid of
    -1 : Writeln('Error en el fork!!!');
    0 : begin {Proceso hijo }
          Writeln('Proceso hijo:');
          fpExecv('/bin/ls', PP);
        end;
    else begin {Proceso padre }
          Writeln('Proceso padre. ');
          fpWaitpid(pid, status, options);
        end;
  end;
```

(12) Modifique el programa anterior para que no haga falta pasar la ruta completa al comando `ls` y guarde la versión modificada como “Prog_0514.pp”. Emplee la documentación de Freepascal para encontrar la función `Exec*` apropiada para ello.

(13) Cree un programa que muestre por pantalla los N primeros términos de la serie de Fibonacci (Siendo N un parámetro tomado de la línea de comandos). Cada termino deberá calcularse en un proceso hijo distinto.

