PROCESOS EN LINUX

Procesos en Linux (Repaso)

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

Pagedaemon:
- Proceso 2
- Soporta la paginación de la memoria

Ingrese 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_0701.pp`**" y compílelo.

```
program Prog_0701;

Uses OldLinux;

begin

    Writeln('Identificador de proceso: ', getpid);
    Writeln('Identificador del proceso padre: ', getppid);

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_0702.pp**" y compílelo.

```
program Prog_0702;

Uses OldLinux;

begin

    Writeln('Identificador de usuario: ', getuid);
    Writeln('Identificador efectivo de usuario: ', geteuid);
    Writeln('Identificador de grupo: ', getgid);
    Writeln('Identificador efectivo de grupo: ', getegid);

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_0702_suid**" y "**prog_0702_sgid**".
Ejecute los comados **chmod u+s prog_0702_suid** y **chmod g+s prog_0702_sgid**.
Ejecute el comando **su** y escriba la contraseña de root. (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_0703.pp**" y compílelo.

```
program Prog_0703;

Uses OldLinux;

begin

    Writeln('El directorio actual es: ',getenv('PWD'));
    Writeln('El Path es: ',getenv('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?
(4) Modifique el programa para que muestre otras variables de entorno y guárdelo como "**prog_0704.pp**" . (Para ver las variables definidas en su entorno, use el comando **set**).

Creación de procesos

Existen dos tipos de procesos:
- Creados por programa: Ej. fork().
- Creados por procesos: Ejecutados por el usuario. Ej: tipear un comando.

Ambos son hijos del proceso inicial que el sistema operativo lanzó en su inicialización.

Diferencia:
- Procesos creados por programa: Se consideran hijos del proceso que desencadenó su creación.
- Procesos creados por un usuario: Pueden considerarse padres independientes

Para crear un proceso desde un programa se utiliza la llamada al sistema fork:
- Permite crear procesos por programa.
- 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.

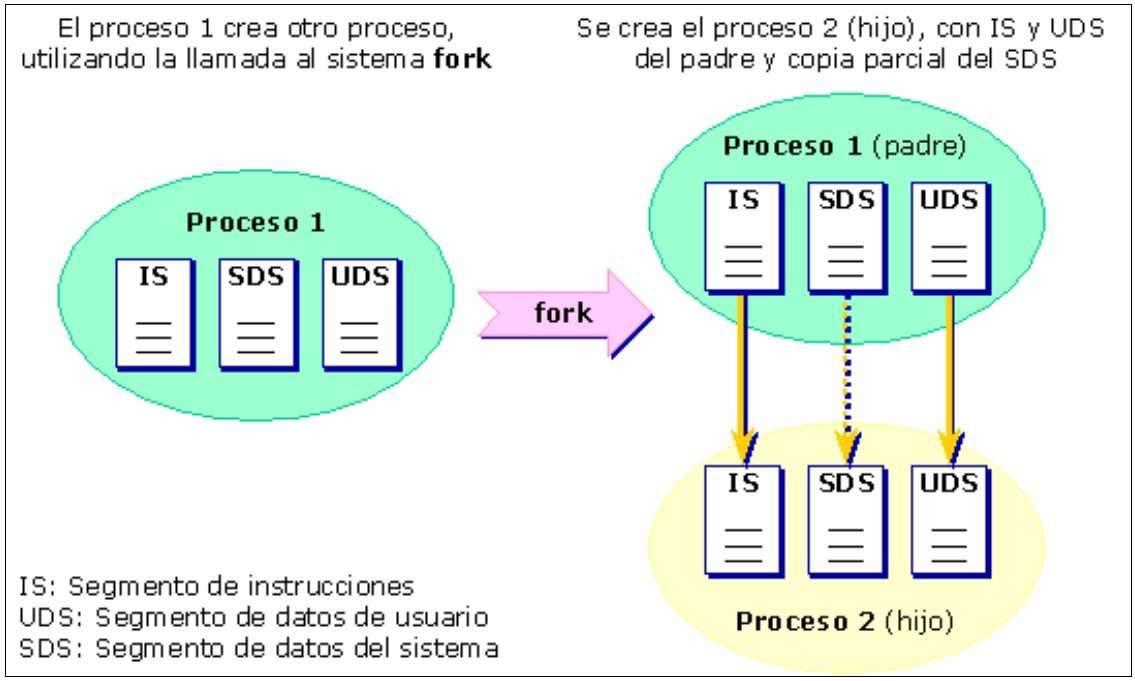
Esta función tiene tres posibles valores de retorno:
- Igual 0: Este código lo **recibe el proceso hijo.**
- Mayor que 0: Este código lo **recibe el proceso** que ha ejecutado la llamada al sistema, es decir el proceso **padre**. Y su **valor** es el número de proceso que **identifica al proceso hijo** 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 getpid()

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 getppid()

Razones por las que puede fallar la creación de procesos, pueden ser por:
- Existir muchos procesos en el sistema
- Exceder el número de procesos para el usuario real(CHILD_MAX).

No se sabe que proceso comienza a ejecutarse primero, si el padre o el hijo.
- Esto depende del algoritmo de scheduling usado por el kernel.
- Para garantizar el comienzo en un determinado orden se puede utilizar la función de demora o por medio de señales.



Creación de procesos con fork().

El proceso hijo hereda del padre:

- Entorno
- Permisos
- Control de la terminal
- Archivos abiertos
- Límites de los recursos

Las diferencias entre ellos son:

- Valor devuelto por fork()
- Process ID
- Diferentes padres

Escriba el siguiente programa y nómbrelo "**prog_0705.pp**" y compílelo.

```
program Prog_0705;

Uses OldLinux;

var
    pid: Longint;

begin
    pid := Fork;

    case pid of
      -1 : Writeln('Error en el fork!!!');
      0  : begin
            Writeln('Soy el hijo:');
            Writeln(' Id del proceso: ', getpid);
            Writeln(' Id de mi proceso padre: ', getppid);
           end;
      else begin
            Writeln('Soy el padre:');
            Writeln(' Id del proceso: ', getpid);
            Writeln(' Id de mi proceso padre: ', getppid);
           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_0706.pp**" y compílelo.

```
program Prog_0706;

Uses OldLinux, Crt;

var
    pid: Longint;

begin
    pid := Fork;

    case pid of
      -1 : Writeln('Error en el fork!!!');
      0  : begin
            Writeln('Soy el hijo, y mi PID es: ', getpid);
            Delay(50000);
           end;
      else begin
            Writeln('Soy el padre, y mi PID es: ', getpid);
           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 **`ps -f`** y observe los resultados. Quién es el padre del proceso hijo? Quién debería serlo? Por qué ocurre esto?

Escriba el siguiente programa y nómbrelo "**prog_0707.pp**" y compílelo.

```
program Prog_0707;

Uses OldLinux;

var
    pid: Longint;
    a: Integer;

begin
    a:=5;
    pid := Fork;

    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_0708.pp**" y compílelo.

```
program Prog_0708;

Uses OldLinux;

var
    pid: Longint;
    i, n: Integer;

begin
    n := 5;

    for i:=1 to n do
      begin
          pid := Fork;
          if pid <> 0 then break;
      end;
    Writeln('El padre del proceso ', getpid, ' es: ', getppid);

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 este versión modificada como "**prog_0709.pp**". Qué modificaciones requirío el programa para comportarse de esta manera? Por qué?

Escriba el siguiente programa y nómbrelo "**prog_0710.pp**" y compílelo.

```
program Prog_0710;

Uses OldLinux;

var
    pid, options: Longint;
    status: ^Integer;

begin

    pid := Fork;
    status := 0;
    options := 0;

    case pid of
      -1 : Writeln('Error en el fork!!!');
      0  : begin  {Proceso hijo }
             Writeln('Proceso hijo:');
             Execl('/usr/bin/ls -l');
           end;
      else begin  {Proceso padre }
             Writeln('Proceso padre.');
             Waitpid(pid, status, options);
           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?

(10) Modifique el programa anterior para que no haga falta pasar la ruta completa al comando **ls** y guarde la versión modficada como "**prog_0711.pp**". Emplee la documentación de Freepascal para encontrar la función Exec* apropiada para ello. (Dentro del manual de referencia de las unidades "**rtl.pdf**").

Escriba el siguiente programa y nómbrelo "**prog_0712.pp**" y compílelo.

```
program Prog_0712;

uses OldLinux;

var
    pid: Longint;

begin

    pid := Fork;

    case pid of
      -1 : Writeln('Error en el fork!!!');
      0  : begin  {Proceso hijo }
             Writeln('PID antes del EXEC: ', getpid);
             Writeln('PPID antes del EXEC: ', getppid);
             Writeln('Ejecutando EXEC...');
             Execl('./prog_0701');
           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_0701**?

Escriba el siguiente programa y nómbrelo "**prog_0713.pp**" y compílelo.

```
program Prog_0713;

Uses OldLinux;

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

var
    pid, options: Longint;
    status: ^Integer;
    PP: PPchar;

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

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

    pid := Fork;
    status := 0;
    options := 0;

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

(12) Modifique el programa anterior para que no haga falta pasar la ruta completa al comando **ls** y guarde la versión modficada como "**prog_0714.pp**". Emplee la documentación de Freepascal para encontrar la función Exec* apropiada para ello.

Confeccione un informe **original**, **conciso** y **completo** donde se de respuesta a las preguntas y consignas precedidas por un número encerrado entre paréntesis. Éste informe deberá ser confeccionado y entregado por cada grupo que llevó a cabo las actividades. La fecha límite de entrega es el **16/10/2007**.
La longitud máxima del informe es de tres páginas (sin contar las líneas correspondientes a código fuente).