Programa en segundo plano para Linux

Profesor: Victor Valenzuela

Consideraciones para correr un programa en Perl

- #!/usr/bin/perl -> es necesario al comienzo del archivo, para que se reconozca el lenguaje utilizado en el script

- Perl algo.pl -> correr el programa usando perl directamente para interpretar.

- Otra alternativa es cambiar el código fuente a ejecutable y luego ejecutarlo directamente:
  - Chmod +x algo.pl -> algo.pl se convierte en ejecutable.
  - ./algo.pl -> ejecuta directamente algo.pl

Correr un programa en Background (desde Perl)

- Utilizando la función system (que ejecuta comandos de consola (también existe en C)) podemos ejecutar el programa requerido en background, usando el simbolo “&”.

- Ejemplo
  - system(/usr/bin/perl algo.pl &);

- Esto sucede porque “&” al final de un programa ejecutado en consola, lo fuerza a ejecutarse en background.

Respecto a otros lenguajes y más ideas…

- Kill a través de Perl:
- https://www.tutorialspoint.com/perl/perl_kill.htm
- Correr un programa en Python:
- https://www.journaldev.com/16140/python-system-command-os-subprocess-call
- Kill a través de Python:
- https://www.programcreek.com/python/example/221/os.kill
- Correr un programa en c:
- https://www.tutorialspoint.com/c_standard_library/c_function_system.htm
- Kill a través de c:
- https://linux.die.net/man/2/kill

Mini fork en perl

```
use strict;
use warnings;
use 5.010;

say "PID $$";
my $pid = fork();
die if not defined $pid;
say "PID $$ ($pid)";
```

****** Resultado *****

```
PID 63778
PID 63778 (63779)
PID 63779 (0)
```

Utilización de fork()

- Fork crea un proceso hijo, que queda supeditado al proceso padre (el que utiliza la función) y que es una copia de padre (por lo que ejecuta el mismo código desde la ejecución del fork).
- Cuando se ejecuta:
- my $pid = fork();
- Este retorna:
  - Si es el padre, $pid sera el PID del hijo.
  - Si es el hijo, $pid sera 0.
  - Si no se puede hacer fork por falta de recursos, $pid será no definido (undefined).

Ejemplo1:padre e hijo

```
#!/usr/bin/perl
my $pid = fork();
if (not defined $pid) {
print “no hay recursos disponibles.\n”;
} elsif ($pid == 0) {
print “SOY EL HIJO\n”;
sleep 5;
print “SOY EL HIJO2\n”;
exit(0);
} else {
print “SOY EL PADRE\n”;
waitpid($pid,0);
}
print “AQUI VIENE CHUCK NORRIS!!!!\n”;
```

Resultado

- $ ./ejemplo.pl
  SOY EL HIJO
  SOY EL PADRE
  - espera por 5 segundos -
  SOY EL HIJO2
  AQUI VIENE CHUCK NORRIS!!!!

Explicación

- El comando waitpid() simplemente espera a que el proceso hijo realice un exit(0) y termine. Si no se hiciera esto en el ejemplo, el hijo se convertiría en un proceso zombie (difunto), lo que significa que se desliga de su proceso padre.
- Así que esto es exactamente lo que sucede cuando este programa se ejecuta:
  - El programa realiza un fork, ahora existen 2 procesos, uno es el hijo y el otro es el padre.
  - **if (not defined $pid)** hace que el programa se termine si no hay recursos disponibles.
  - **elsif ($pid == 0)** es la condición que indica que es el hijo, por lo que imprimiría "SOY EL HIJO", esperaría por 5 segundos y luego imprimiría por panralla "SOY EL HIJO2" y realizaría salida del programa con exit(0);
  - Mientras lo anterior está corriendo en el proceso hijo, el padre está ejecutando el bloque de código **else {}** que imprime "SOY EL PADRE" y luego espera a que el proceso hijo termine.

Ejemplo 2: múltiples hijos

```
#!/usr/bin/perl
@array = qw(presente!!!!);
$num = “10”;
for(1..$num) {
my $pid = fork();
if ($pid) {
# padre
push(@childs, $pid);
} elsif ($pid == 0) {
# hijo
print “@array\n\n”;
sleep 5;
exit(0);
} else {
die “no se pudo hacer fork: $!\n”;
}
print “ANTES DEL FINAL DE FOR\n”;
}
print “ESPERANDO A QUE LOS HIJOS TERMINEN…\n\n”;
foreach (@childs) {
waitpid($_, 0);
}
print “POR FIN TODO TERMINÓ…”;
```

Explicación

- Esto hace que desde el 1 al 10, se realice un fork,
- Si es el padre añade $pid (pid del hijo) al array @childs,
- Si es un hijo imprime @array espera por 5 segundos y luego termina.
- Luego, al final, se dice que por cada pid contenido en @childs, el proceso padre espera a que se termine cada uno de los procesos hijos guardados.
- En este programa tendriamos 11 procesos, 1 padre y 10 hijos.

Ejemplo fork en c

```c
#include <stdio.h>
#include <unistd.h>
int main(int argc, char **argv)
{
    printf("--beginning of program\n");
    int counter = 0;
  pid_t pid = fork();
  if (pid == 0)
    {
    // child process
    int i = 0;
    for (; i < 5; ++i)
        {
            printf("child process: counter=%d\n", ++counter);
        }
    }
  else if (pid > 0)
    {
    // parent process
    int j = 0;
    for (; j < 5; ++j)
        {
            printf("parent process: counter=%d\n", ++counter);
        }
    }
  else
    {
    // fork failed
        printf("fork() failed!\n");
    return 1;
    }
    printf("--end of program--\n");
  return 0;
}
```

Ejemplo fork en python

```
import os

def child():
   print('\nA new child ',  os.getpid())
   os._exit(0)

def parent():
   while True:
      newpid = os.fork()
      if newpid == 0:
         child()
      else:
         pids = (os.getpid(), newpid)
         print("parent: %d, child: %d\n" % pids)
      reply = input("q for quit / c for new fork")
      if reply == 'c':
          continue
      else:
          break

parent()
```

Ejercicios

- Cree una aplicación que cree un proceso hijo que imprima en pantalla cada 2 segundos una frase. Haga que el proceso padre mate a este proceso hijo, pasados 10 segundos.
- Cree una aplicación que cree un proceso hijo que guarde dentro de un archivo la información de los clientes que están conectados al servidor actual cada 5 segundos, por un periodo de 30 segundos. Luego el proceso padre lo termina.

Ejercicios

- Cree un programa que cree 5 procesos hijos que escriban por pantalla un nombre distinto para cada uno de ellos (con que tenga alguna diferencia es valido), luego de lo cual deben esperar por 3 segundos y despedirse, antes de terminar. El proceso padre debe finalizar después de que todos los procesos hijos se finalizan y debe avisar cada vez que inicie un proceso hijo.