Escuela Politécnica Superior de Elche

SISTEMAS INFORMÁTICOS EN TIEMPO REAL

2º Ingeniería Industrial

CREACIÓN Y MANEJO BÁSICO DE PROCESOS

Luis Miguel Jiménez
Rafael Puerto

Departamento de Ingeniería de Sistemas Industriales
Área de Ingeniería de Sistemas y Automática

© ISA-UMH

OBJETIVOS

- Familiarizarse con el entorno de desarrollo en SUN-Solarix /Linux
- Utilizar un editor de textos para escribir programas, compilar y ejecutar de forma remota.
- Repasar los conceptos de creación y gestión básica de procesos vistos en teoría
- Realizar los ejercicios propuestos a continuación

Ejercicio 1

Escribir, compilar y ejecutar los siguientes programas que se muestran a continuación. Describe brevemente qué tarea realizan y cómo la realizan.

Nota: El código de los ejemplos propuestos puede descargarse de la página web de la asignatura

Programa 1: Creación de procesos

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

int main (void)
{
  pid_t pid;

  pid = fork();
  switch (pid)
  {
      case -1:    perror ("No se ha podido crear el hijo");
                  break;
      case 0:     printf("Soy el hijo, mi PID es %d y mi PPID es
                  %d\n", getpid(), getppid());
                  break;
      default:    printf ("Soy el padre, mi PID es %d y el PID de mi
                           hijo es %d\n", getpid(), pid);
  }
  exit(0);
}
```

Programa 2: Espera la terminación de un proceso (wait)

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


int main(void)
{
  pid_t childpid, childdead;
  int i;

  childpid = fork();
  if (childpid == -1)
  {
      perror("fork no pudo crear el hijo");
      exit(1);
  }
  else if (childpid == 0)
  {
       printf ("Soy el hijo (PID %d) y mi padre es (PID %d), voy a contar
hasta 100000 \n", getpid(), getppid());
       for (i=0;i<100000; i++) {}
  }
  else
  {
       printf("Soy el padre (PID %d) y voy a esperar a mi hijo (PID %d)\n",
                              getpid(),childpid);
       if ((childdead = wait(0))== -1)
           perror("No he podido esperar al hijo");
       else
           printf("Mi hijo con pid %d , ha muerto\n",childdead);
  }
  exit(0);
}
```


Programa 3:

- **Cambio de la imagen de un proceso**
- **Paso de parámetros al programa (argv)**
- **Paso de parámetros al nuevo proceso**
- **Ejemplo de uso del programa:** **`p3 ls -la`**

```
/* Ejemplo de uso de la función  execvp */

#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/wait.h>

int main (int argc, char *argv[])
{
  pid_t childpid;

  childpid = fork();

  if (childpid == -1)
  {
      perror("Error al crear el hijo");
      exit(1);
  }
  else if (childpid ==0)
  {
       if(argc>1)
       {
           printf("Soy el hijo y voy a ejecutar el comando %s\n",
                                argv[1]);
           if(execvp(argv[1], &argv[1])<0)
           {
                perror("Error al ejecutar el comando");
                exit (1);
           }
       } else
           printf("Soy el hijo y no has escrito ningún comando\n");
  }
  else
  {
      printf("Soy el padre, espero al hijo y termino\n");
      wait(0);
  }

  exit(0);
}
```

Ejercicio 2: procesos zombies

Realizar un programa en el que se cree un hijo. El padre debe imprimir su PID y el de su hijo y salir inmediatamente. El hijo mostrará su PID y el de su padre esperará 6 segundos (consultar el manual en línea para el comando ***sleep***) y a continuación escribirá su PID y el de su padre. ¿Qué diferencias se aprecian entre la salida por pantalla del padre y del hijo? ¿A que piensas que es debido?

Ejercicio 3:

Implementar una aplicación concurrente que calcule el cuadrado de los 20 primeros números naturales y almacene el resultado, repartiendo la tarea entre dos procesos:

- Crear dos procesos Hijos:
  - 1er Proceso Hijo: Realiza la operación sobre los números impares. Almacena el resultado en un fichero de texto (impares.dat)
  - 2º: Proceso Hijo Realiza la operación sobre los números pares. Almacena el resultado en un fichero de texto (pares.dat)

- El proceso padre espera la terminación de los hijos, '***obtiene***' el resultado de cada hijo y muestra los valores ordenados en pantalla
- La comunicación entre procesos se realizará mediante ficheros de **texto**, donde se almacenará cada valor en una línea de texto.