Programación Avanzada

Curso 2023/2024

prof. Claudio Rossi

Practica 1: procesos y fork

Objetivos:
1. recuperar datos de los proceso (usuario, ID, ...)
2. funciones *fork* y *execve*.

Comandos de S.O.: ps, top (elencan proceso del sistema)

Funciones utilizadas:

Parte1

- uid_t *getuid()*: proporciona el identificador ID de usuario del proceso

- struct passwd **getpwuid(uid_t uid)*: proporciona un puntero a una estructura que contiene información sobre el usuario *uid*:

```
struct passwd{
        char *pw_name;
        char *pw_passwd;
        uid_t pw_uid;
        ...
}
```

- struct group **getgrgid(gid_t gid)*: proporciona un puntero a una estructura que contiene información sobre el grupo de usuario *gid* (Identificador ID de grupo para dicho usuario):

```
struct group {
char gr_name;
gid_t gr_gid;
char **gr_mem;
 }
```

- pid_t *getpid(),* pid_t *getppid():* retornan el PID del proceso y del padre
- void *exit*(int status): termina el proceso y retorna *status* al padre
    - tipicamente, 0=ok, -1=error. Uso de constantes: EXIT_SUCCESS, EXIT_FAILURE

- void *perror*(const char *message): imprime un mensaje de error asociado a la cadena que tiene como argumento

Parte 2

- pid_t fork(): creación de un nuevo proceso, si la operación se realiza de forma satisfactoria, la función fork devuelve al proceso padre el identificador ID del proceso hijo, y al proceso hijo un 0; en caso contrario, se devuelve –1 al proceso padre

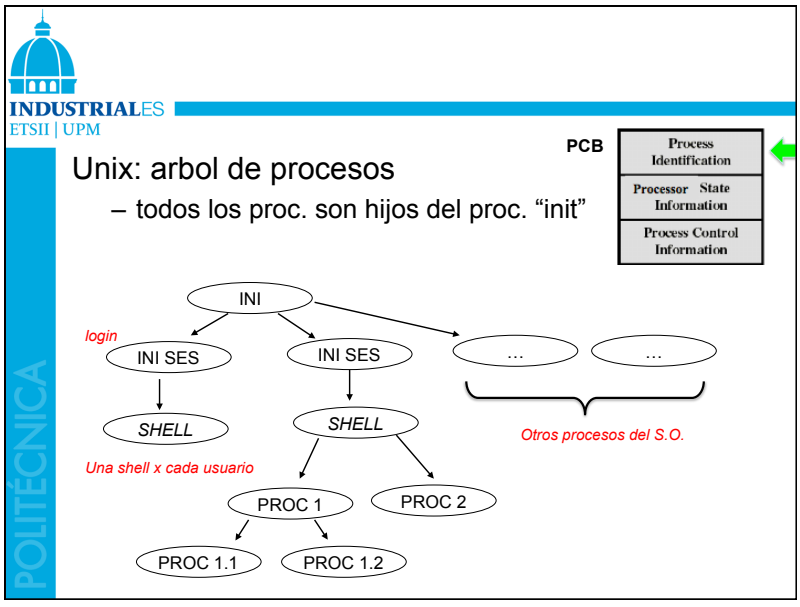
- pid_t wait(int *status): provoca una espera del proceso padre hasta que cualquier proceso hijo finalice. La información del estado del proceso hijo se almacena en status

- pid_t waitpid(pid_t pid, int *status, int options): provoca una espera del proceso padre hasta que el proceso hijo con identificador pid finalice. Si pid es –1 realiza una espera asociada a la finalización de todos los procesos hijos, es decir, se comporta como wait(). La información del estado del proceso hijo se almacena en status. El tercer argumento es una máscara de bits que sirve para modificar el funcionamiento

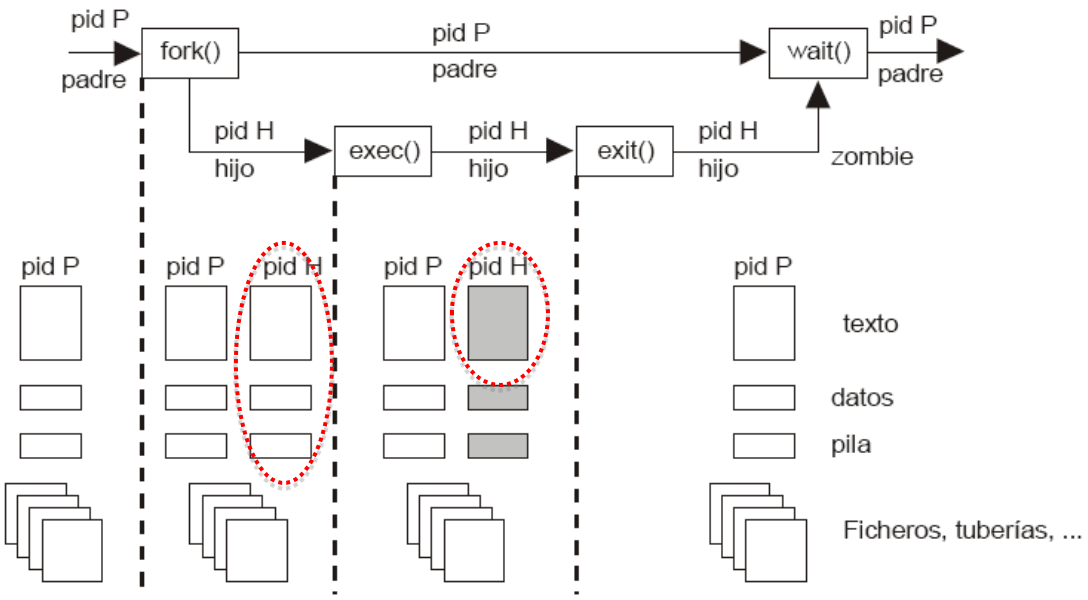
- int execve(...): permite la ejecución del programa nombre como un nuevo proceso imagen. La lista de argumentos es:
    - const char *filename: contiene el nombre del programa ejecutable.
    - char *const argv[]: el primer elemento de array es el nombre de programa, y el último un puntero NULL.
    - char *const env[]: permite declarar el posible entorno de funcionamiento delprograma

Parte 1.
El S.O. mantiene una tabla de procesos, con varia información sobre ellos. Esta información puede ser conocida por los procesos. a través de las funciones mencionadas arriba.

Parte 2.
Recordar que en unix todos los procesos son generados a partir de un proceso común:



Para generar un nuevo proceso, se usa la función fork, que causa la clonación del proceso, generando un programa "hijo". Este podrá cambiar su código, ejecutando el de otro proceso. Para ello, se usa la función "exec" (o su variantes, p.e. "execve").



- Tras la fork, el hijo tiene el mismo texto (código) del padre, y comparte todos los recursos, pero NO la memoria (datos, pila) !
- Tras la exec, el hijo cambia su segmento texto. Se transforma de hecho en otro proceso.

Programa 1

```
/* Este programa muestra información del usuario que lo está
ejecutando */
#include <grp.h>
#include <pwd.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>


/* Función principal */

int main(void)
{
  /* Definición de las variables utilizadas en el programa */

  uid_t yo; /* uid_t es un entero que representa al usuario IDs */
  struct passwd *mipass;      /* passwd es una estructura que
                                 presenta distintos campos
                                 con información del usuario */
  struct group *migrup; /* group es una estructura que
                           presenta distintos campos
                           con información del grupo de usuarios */
  char **miembros; /*   Array de    punteros    con    los    nombres
                           de los usuarios del grupo */
```

```
  pid_t PID,PPID;


  /* Información del proceso */
  PID  = getpid();
  PPID = getppid();


  printf("El ID del proceso es %d, el de su padre es %d \n",
             PID,PPID);
  printf("(Comprobad,  a  traves  del  comando  ps,  que  %d  es
efectivamente el PID de la consola!)\n",PPID);



  yo = getuid();

  mipass = getpwuid(yo);
  if(!mipass)
      /* contiene el login del usuario */
      /* palabra clave encriptada del usuario */
      /* Idinetificador ID de usuario */
      {
        printf("No encuentro al usuario %d \n",(int) yo);
        exit(EXIT_FAILURE);
      }

  /* Salida por pantalla de la información de usuario */
  printf("Soy : %s \n",mipass->pw_gecos);
  printf("Mi login es: %s\n", mipass->pw_name);
  printf("Mi id es: %d \n", (int) (mipass->pw_uid));
  printf("Mi directorio de trabajo es: %s\n", mipass->pw_dir);
  printf("Mi shell es: %s\n", mipass->pw_shell);


  migrup = getgrgid(mipass->pw_gid);
  if(!migrup)
      {
        printf("No encuentro el grupo %d \n",(int) (mipass->pw_gid));
        exit(EXIT_FAILURE);
      }

  /* Salida por pantalla de la información de grupo de usuarios */
  printf("Mi grupo es: %s (%d)\n",
       migrup->gr_name, (int) (mipass->pw_gid));
  printf("Los miembros del grupo son:\n");
  miembros = migrup->gr_mem;


  while(*miembros)
      {
        printf("%s \n",*(miembros));
        miembros++;
      }

  exit(EXIT_SUCCESS);
}
```

Programa 2

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


/* Variables globales */

pid_t pid;  /* pid_t es un entero que representa el identificador
                   (ID) de un proceso */


/* Función principal */
int main (void)
{
  /* Creación de un proceso hijo */
  switch(pid = fork())
      {
      case (pid_t) -1:
        perror("fork");
        exit(-1);
      case (pid_t) 0:
        printf("Hola, yo soy el hijo, y tengo PID %d. Mi padre es el
PID %d\n",getpid(),getppid());
        break;
      default:
        printf("Hola, yo soy el padre, y tengo PID %d\n",getpid());
        break;
      }

  exit(0);
}
```

Programa 3

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>


/* Variables globales */



/* Función principal */
int main (void)
{
  int i;
  int status;
  pid_t pid;      /*   pid_t   es   un   entero   que   representa   el
identificador ID de un
                     proceso */

  /* Creación de un proceso hijo */
  switch(pid = fork())
      {
      case (pid_t) -1:
        /* void perror(const char *message): imprime un
```

```
            mensaje de error asociado a la cadena que tiene como
argumento */
        perror("fork");
        exit(-1);
      case (pid_t) 0:
        printf("Hola, yo soy el hijo, y tengo PID %d. Mi padre es el
PID %d\n",getpid(),getppid());
        for(i=0;i<20;i++)
            printf("Soy   el   proceso   hijo   y   estoy   perdiendo
tiempo...\n");
        break;
      default:
        printf("Hola,  yo  soy  el  padre,  y  tengo  PID  %d.  Estoy
esperando a que mi hijo termine...\n",getpid());



        waitpid(pid,&status,0);

        /* Alternativa: la función  wait(int *status) provoca
             una espera del proceso padre hasta que cualquier
             proceso hijo finalice. La información del estado del
             proceso hijo se almacena en status.*/


        printf("Padre: mi hijo ya termino'!\n");
        break;
      }

  exit(0);
}
```

Programa 4

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


/* Variables globales */



/* Función principal */
int main (void)
{
  int i;
  int status;
  pid_t pid;      /*  pid_t  es  un  entero  que  representa  el
identificador ID de un
                     proceso */

  char *nombre = "programa_hijo";
  // probar con otros !
  // char *nombre = "/bin/ls";
  // char *nombre = "/bin/ps";
```

```
  char *av[2];  /* lista de argumentos */

  /* Creación de un proceso hijo */
  switch(pid = fork())
      {
      case (pid_t) -1:
        /* void perror(const char *message): imprime un
             mensaje de error asociado a la cadena que tiene como
argumento */
        perror("fork");
        exit(-1);
      case (pid_t) 0:
        printf("Hola, yo soy el hijo, y tengo PID %d. Mi padre es el
PID %d\n",getpid(),getppid());

        printf("ahora ejecuto otro programa !\n");
        av[0]=nombre;
        av[1]=NULL;

        if(execve(nombre,av,NULL) == -1)
            {
              perror("execve");
              exit(EXIT_FAILURE);
            }

        break;
      default:
        printf("Hola, yo soy el padre, y tengo PID %d.\n",getpid());


        break;
      }

  exit(0);
}
```

Programa HIJO para el programa 4

```
#include <stdlib.h>
#include <stdio.h>

// El ejecutable de este programa se tiene que
// llamar como especificado en el programa4!

/* Función principal */
int main(void)
{
  int i=0;

  /* Introducción por teclado del número de veces que se repetirá el
bucle while */


  for(i=0;i<10;i++)
      printf("Hola soy el nuevo proceso hijo !\n");


  exit(EXIT_SUCCESS);
}
```

Programa 5

Escribir un programa que crea DOS hijos. El programa deberá tener una variable Z. Cada uno de los hijos deberá cambiar el valor de Z y imprimirlo en pantalla (usar valores distinos!). Se comprobará así que el segmento TEXTO se clona, pero el segmento DATOS es distinto!