Universidad
Carlos III de Madrid

SISTEMAS INFORMÁTICOS EN TIEMPO REAL

PRÁCTICAS DE LABORATORIO

DPTO. DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

José María Armingol Moreno
Miguel Ángel de Miguel Paraíso

PRÁCTICA 2

PROCESOS

Los procesos se conforman por programas cuya ejecución es administrada por el sistema operativo. Esto involucra, la identificación de cada proceso (ID de proceso), los mensajes de control o señales, los recursos disponibles para el proceso, comunicación entre procesos y otros. La comunicación entre procesos (IPC, siglas del nombre en inglés, Inter-Process Communication) permite garantizar la interacción incluso con programas ya creados en formato ejecutable, y desde leguajes de programación diferentes.

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;
    /* Información del usuario */
    /* uid_t getuid(): proporciona el identificador ID de usuario del proceso */
    yo = getuid();
    /* struct passwd * getpwuid(uid_t uid): proporciona un puntero a una estructura que contiene
    información sobre el usuario uid:
        struct passwd{
            char *pw_name; //contiene el login del usuario
            char *pw_passwd; // palabra clave encriptada del usuario
            uid_t pw_uid; //Idinetificador ID de usuario
            .....
            } */
    mipass = getpwuid(yo);
    if(!mipass)
    {
        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);
    /* Información del grupo */
    /* 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; // nombre del grupo de usuarios
            gid_t gr_gid; // identificador ID del grupo
            char **gr_mem; // array de punteros con los nombres de todos los usuarios del grupo
            }
    */
```

```
    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;
    /* Array de punteros con los nombres de los usuarios del grupo */
    while(*miembros)
    {
        printf("%s \n",*(miembros));
        miembros++;
    }
    return EXIT_SUCCESS;
}
/* Fin función principal */
```

PROGRAMA 2

```
/* Creación, terminación y control de procesos hijo a partir de un proceso padre */
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
/* Prototipos */
void hijo(void);
void padre(void);
/* Variables globales */
pid_t pid;
/* pid_t es un entero que representa el identificador ID de un proceso*/
/* Función principal */
int main (void)
{
    /* 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 */
    /* 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(EXIT_FAILURE);
        case (pid_t) 0:
            hijo();
            break;
        default:
            padre();
            break;
    }
    /* void exit(int status): esta función termina el proceso con estado status.
    El valor de status es un entero comprendido entre 0 y 255. Normalmente se emplea el valor 0
    para indicar salida correcta (SUCCESS) y 1 salida por fallo (FAILURE) */
    exit(EXIT_SUCCESS);
```

```
}
    /* Fin función principal */


    /* Declaración de las dos funciones que permiten identificar ambos procesos */
void hijo(void)
{
    printf("Hola soy el proceso hijo \n");
}

void padre(void)
{
    printf("Hola soy el proceso padre \n");
}
```

PROGRAMA 3

```
/* Creación, terminación y control de procesos hijo a partir de un proceso padre */
# include <sys/types.h>
# include <sys/wait.h>
# include <unistd.h>
# include <stdlib.h>
# include <stdio.h>
/* Prototipos */
int hijo(void);
void padre(void);
/* Función principal */
int main (void)
{
    pid_t pid;
    int status;
    /* Creación de un proceso hijo */
    switch(pid = fork())
    {
        case (pid_t) -1:
            perror("fork");
            exit(EXIT_FAILURE);
        case (pid_t) 0:
            exit(hijo());
            break;
        default:
            /* 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 (deberá valer
            0 “OR”). La función devuelve el identificador pid del proceso cuyo estado está
            proporcionando. */
            padre();
            /* 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 (deberá valer
            0 “OR”). La función devuelve el identificador pid del proceso cuyo estado está
```

```
            proporcionando. */
            waitpid(-1, &status, 0);
            /* 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.*/
            //wait(&status);
            printf("Ya ha terminado el proceso hijo\n");
            break;
    }

    exit(EXIT_SUCCESS);
}
/* Fin función principal */
/* Declaración de las dos funciones que permiten identificar ambos procesos */
int hijo(void)
{
    printf("Hola soy el proceso hijo\n");
    return 1;
}

void padre(void)
{
    printf("Hola soy el proceso padre\n");
}
```

PROGRAMA 4

```
/* Creación, terminación y control de procesos hijo a partir de un proceso padre */
# include <sys/types.h>
# include <sys/wait.h>
# include <unistd.h>
# include <stdlib.h>
# include <stdio.h>
/* Prototipos */
int hijo(void);
void padre(void);
/* Función principal */
int main (void)
{
    pid_t pid;
    int status;
    /* Creación de un proceso hijo */
    switch(pid = fork())
    {
        case (pid_t) -1:
            perror("fork");
            exit(EXIT_FAILURE);
        case (pid_t) 0:
            exit(hijo());
            break;
        default:
            /* 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 (deberá valer
            0 “OR”). La función devuelve el identificador pid del proceso cuyo estado está
```

```
            proporcionando. */
            padre();
            /* 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 (deberá valer
            0 “OR”). La función devuelve el identificador pid del proceso cuyo estado está
            proporcionando. */
            waitpid(-1, &status, 0);
            /* 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.*/
            //wait(&status);
            printf("Ya ha terminado el proceso hijo\n");
            break;
    }

    exit(EXIT_SUCCESS);
}
/* Fin función principal */
/* Declaración de las dos funciones que permiten identificar ambos procesos */
int hijo(void)
{
    sleep(1);
    printf("Hola soy el proceso hijo\n");
    return 1;
}

void padre(void)
{
    printf("Hola soy el proceso padre\n");
}
```

PROGRAMA 5

Programa padre

```
///Programa padre
/* Creación, terminación y control de procesos hijo a partir de un proceso padre */
# include <sys/types.h>
# include <sys/wait.h>
# include <unistd.h>
# include <stdlib.h>
# include <stdio.h>
char *nombre = "hijo";
/* Función principal */
int main(void)
{
    pid_t pid;
    char *av[2];
    int status;
    /* Creación de un proceso hijo */
    switch(pid = fork())
    {
        case (pid_t) -1:
```

```
            perror("fork");
            exit(EXIT_FAILURE);
        case (pid_t) 0:
            /* int execve(const char *filename, char *const argv[], char *const envp[]):
            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 del
            programa */
            av[0]=nombre;
            av[1]=NULL;
            if(execve(nombre,av,NULL) == -1)
            {
                perror("execve");
                exit(EXIT_FAILURE);
            }
    }
    printf("Soy el proceso padre\n");
    /* Espera del proceso padre a la terminación del proceso hijo */
    wait(&status);
    printf("Fin del proceso hijo\n");
    exit(EXIT_SUCCESS);
}
/* Fin función principal */
```

Programa hijo

```
///Programa hijo
/* Programa hijo.c, cuyo ejecutable se lanza desde el proceso padre */
#include <stdlib.h>
#include <stdio.h>
/* Función principal */
int main(void)
{
    int i = 0;
    int j = 0;
    char c;
    /* Introducción por teclado del número de veces que se repetirá el bucle while */
    printf("Número de ciclos <1-20>:");
    fflush(stdout);
    scanf("%s",&c);
    i = atoi(&c);
    while(j < i)
    {
        printf("Hola soy de nuevo el proceso hijo %d\n",j);
        j++;
    }
    exit(EXIT_SUCCESS);
    printf("\n");
}
/* Fin función principal */
```

Problemas Propuestos:

1. Modificar p2-4.c para crear otro proceso hijo que ejecute otra función hijo2 con un sleep de un tiempo mayor. El proceso padre debe esperar a que los dos procesos hijos terminen.
2. Modificar los dos códigos (padre e hijo) para que el número de ciclos se pase por terminal como parámetro al padre, y de padre a hijo.