Sistemas Operativos

Formulario de auto-evaluación

Modulo 2. Sesión 3. Llamadas al sistema para el Control de Procesos

Nombre y apellidos:

Ana Alicia Vílchez Ceballos

a) Cuestionario de actitud frente al trabajo.

El tiempo que he dedicado a la preparación de la sesión antes de asistir al laboratorio ha sido de …… minutos.

1. He resuelto todas las dudas que tenía antes de iniciar la sesión de prácticas: . En caso de haber contestado “no”, indica los motivos por los que no las has resuelto:

2. Tengo que trabajar algo más los conceptos sobre:

3. Comentarios y sugerencias:

b) Cuestionario de conocimientos adquiridos.

Mi solución al **ejercicio 1** ha sido:

```
// Ana Alicia Vílchez Ceballos, ejer1.c

// Trabajo con llamadas al sistema del Subsistema de Procesos "POSIX 2.10 compliant"

// precondición: el argumento que le pasamos debe ser un numero entero

#include <sys/types.h>     //Primitive system data types for abstraction of implementation-dependent data types.

#include <unistd.h>                    //POSIX Standard: 2.10 Symbolic Constants        <unistd.h>

#include <stdio.h>

#include <errno.h>

#include <stdlib.h>

int main(int argc, char *argv[])

{

        int var;

        pid_t pid;


        if(argc < 2){

                perror("\nError en el numero de argumentos");

                exit(-2);

        }


        int num = atoi(argv[1]);


        if( (pid=fork()) < 0) {

                perror("\nError en el fork");

                exit(-1);

        }


        if(pid == 0){ // trabajo trealizado por el proceso hijo

                if(num % 2 == 0 )

                        printf("el numero %d es PAR (hecho por proceso %d)\n", num, getpid());
```

```
		else
			printf("el numero %d es IMPAR (hecho por proceso %d)\n", num, getpid());
	}

	else{ // en caso es el proceso padre, pues el PID es distinto de 0
		if(num % 4 == 0 )
			printf("el numero %d es divisible por 4 (hecho por proceso %d)\n", num, getpid());
		else
			printf("el numero %d NO es divisible por 4 (hecho por proceso %d)\n", num, getpid());
	}
}
```

Mi solución a la **ejercicio 3** ha sido:

En el bucle primero, donde no se niega a childpid, son los hijos de los hijos los que ejecutan el código.

En el caso del bucle segundo, donde si se niega, serán los procesos padres los que lleven a cabo la ejecución del bucle.

Mi solución a la **ejercicio 4** ha sido:

```
//Ana Alicia Vílchez Ceballos, ejer4.c y ejercicio 5 (que include una modificación del 4)
```

```
//Trabajo con llamadas al sistema del Subsistema de Procesos "POSIX 2.10 compliant"

//Prueba el programa tal y como está. Después, elimina los comentarios (1) y pruebalo de nuevo.



#include <sys/types.h>     //Primitive system data types for abstraction of implementation-dependent data types.

                                                //POSIX Standard: 2.6 Primitive System Data Types <sys/types.h>

#include <unistd.h>                  //POSIX Standard: 2.10 Symbolic Constants        <unistd.h>

#include <stdio.h>

#include <errno.h>

#include <stdlib.h>


// variables globales:

int nprocs = 5;




void finalizar(pid_t *pid){


// Con este código llevamos a cabo la modicifación propuesta en el ejercicio 5, donde se espera primero a los

// hijos creados en orden impar y luego amlos que tienen orden par

/*

        for (int i=0; i<nprocs; i++) {

                if(i %2 != 0){

                        if (waitpid(pid[i],0,0) > 0) {    // si la espera ha tenido exito y finaliza el hijo tal...

                        printf("Acaba de finalizar mi hijo %d (orden %d)\n",pid[i], i);

                        printf("Solo me quedan %d hijos vivos\n",nprocs - i -1);

                }

        }

        }


        for (int i=0; i<nprocs; i++) {
```

```
                if(i %2 == 0){
                        if (waitpid(pid[i],0,0) > 0) {    // si la espera ha tenido exito y finaliza el hijo tal...
                        printf("Acaba de finalizar mi hijo %d (orden %d)\n",pid[i], i);
                        printf("Solo me quedan %d hijos vivos\n",nprocs - i -1);
                }
        }
        }
*/
        for (int i=0; i<nprocs; i++) {
                if (waitpid(pid[i],0,0) > 0) {   // si la espera ha tenido exito y finaliza el hijo tal...
                printf("Acaba de finalizar mi hijo %d (orden %d)\n",pid[i], i);
                printf("Solo me quedan %d hijos vivos\n",nprocs - i -1);
        }


        }


}
int main(int argc, char *argv[]){
        pid_t pid[nprocs];


        if ((pid[0] = fork())==0) // el proceso padre invoca al hijo
                printf("\nsoy el hijo %d\n", getpid());
        else {


                if((pid[1] = fork()) == 0){
                        printf("\nsoy el hijo %d\n", getpid());// si no es el hijo el proceso padre vuelve a invocar al hijo y continua con el código finalizar(pid)
                }
                else {
                        if((pid[2] = fork()) == 0){
                                printf("\nsoy el hijo %d\n", getpid());// Proceso hijo
```

```
                        }
                        else {
                                if((pid[3] = fork()) == 0){
                                        printf("\nsoy el hijo %d\n", getpid());// Proceso hijo
                                }
                                else {
                                        if((pid[4] = fork()) == 0){
                                                printf("\nsoy el hijo %d\n", getpid());// Proceso hijo
                                        }
                                }
                        }
                }
        }
        finalizar(pid);
}
```

Mi solución a la **ejercicio 6** ha sido:

En este programa lo que llevamos a cabo es la creación de un proceso hijo que se encargará de ejecutar la orden ldd que se encuentra en el directorio

/usr/bin/ y que nos muestra las librerías que necesita un programa, en este caso tarea5

el ultimo argumento será NULL pues siempre va detrás de la lista de parámtros, en este caso el único parámetro que necesita lls es el ejecutable de ./tarea5.

La llamada **`wait`** recibe como parámetro un puntero a entero donde se deposita el valor devuelto por el proceso hijo al terminar; y retorna el PID del hijo. Por tanto el código siguiente a la ejecución de lld no la realiza el hijo creado con fork() sino el padre.