Sistemas Operativos

Formulario de auto-evaluación

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

Nombre y apellidos:

Emilio Chica Jimenez

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: …… (si/no). 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:

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


int main(int argc, char *argv[])
{
	if(argc==2){
		pid_t pid;
		int numero;
		numero = strtol(argv[1],NULL,10);
		if( (pid=fork())<0) {
			perror("\nError en el fork");
			exit(-1);
		}
		else if(pid==0) {  //proceso hijo ejecutando el programa
			if(numero%2==0)
			{
				char buf[]="El numero es PAR\n\n";
				if(write(STDOUT_FILENO,buf,sizeof(buf)+1) != sizeof(buf)+1) {
					perror("\nError en write");
					exit(-1);
				}
				exit(0);
			}
			else
```

```
                {
                        char buf[]="El numero es IMPAR\n\n";
                        if(write(STDOUT_FILENO,buf,sizeof(buf)+1) != sizeof(buf)+1) {
                                perror("\nError en write");
                                exit(-1);
                        }
                        exit(0);
                }
        } else  //proceso padre ejecutando el programa
        {
                if(numero%4==0){
                        char buf[]="El numero es divisible por 4\n\n";
                        if(write(STDOUT_FILENO,buf,sizeof(buf)+1) != sizeof(buf)+1) {
                                perror("\nError en write");
                                exit(-1);
                        }
                }
                else{
                        char buf[]="El numero NO es divisible por 4\n\n";
                        if(write(STDOUT_FILENO,buf,sizeof(buf)+1) != sizeof(buf)+1) {
                                perror("\nError en write");
                                exit(-1);
                        }



                }


        }
}
else{
        printf("\nSintaxis de ejecucion: ./ejercicio1 [<numero>]\n\n");
```

```
                exit(-1);
        }


}
```

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

```
//////////////PARTE 1

Segun se ha comentado está primera parte de la actividad no es nada correcta puesto que al eleminar al padre de los hijos, estos se quedarían “huerfanos” y se irián al padre inicial, es decir el proceso “init” por lo que no se recomienda esta solución.



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


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


        pid_t childpid;
        int nprocs=5;
        int i;
        /*
```

```
	Jerarquía de procesos tipo 1
	*/
	printf("\nJERAQUIA 1\n");
	if(setvbuf(stdout,NULL,_IONBF,0)) {
		perror("\nError en setvbuf");
	}
	for (i=1; i < nprocs; i++) {
		if ((childpid= fork()) == -1) {
			fprintf(stderr, "Could not create child %d: %s\n",i,strerror(errno));
			exit(-1);
		}

		if (childpid){
			printf("Proceso padre: %i\n",getpid());
			exit(0);
		}else
			printf("Proceso hijo: %i\n",getpid());

	}

}

//////////////////PARTE2

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

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

	pid_t childpid;
	int nprocs=10;
	int i;
		/*
	Jerarquía de procesos tipo 2
	*/
	printf("\nJERAQUIA 2\n");
	if(setvbuf(stdout,NULL,_IONBF,0)) {
			perror("\nError en setvbuf");
	}

	for (i=1; i < nprocs; i++) {
		if ((childpid= fork()) == -1) {
			fprintf(stderr, "Could not create child %d: %s\n",i,strerror(errno));
			exit(-1);
		}
		if (!childpid){
			printf("Proceso hijo: %i\n",getppid());
			break;
		}else
			printf("Proceso padre: %i\n",getpid());

	}


}
```

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

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


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


	pid_t valorDevuelto;
	int nhijos=5;
	int nprocess=5;
	int i;
	if(setvbuf(stdout,NULL,_IONBF,0)) {
		perror("\nError en setvbuf");
	}
	for(i=0;i<nprocess;i++){
		if((valorDevuelto=fork())==0){
			printf("Soy el hijo: %i\n\n",getpid());
			exit(0);
		}
	}
	int * status;
```

```
        for (i = 0; i < nprocess; ++i){
                valorDevuelto = wait(&status);
                printf("Acaba de terminar mi hijo con %i\n\n",valorDevuelto);
                printf("Me quedan %i hijos\n\n",--nhijos);
        }

        exit(0);
}
```

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

Primero crea un proceso hijo con fork. Este hijo ejecuta la función execl() que sustituye la imagen del proceso actual por una nueva imagen de proceso. EL proceso que ejecuta se encuentra en el primer argumento de "/usr/bin/ldd" ldd imprime las librerias necesarias por cada programa o comparte librerias especificas en la linea de comandos.

El segundo argumento de execl por convencion es un puntero a el nombre del archivo asociado con el archivo que va a ser ejecutado. El tercer argumento es lo que le vamos a pasar a ldd y por ultimo esta función tiene que acabar en NULL.

El padre espera a que ejecute el hijo el proceso que ha llamado en el execl.

Para obtener el valor de exit del hijo, es decir, el porque ha acabado el proceso hijo del padre hacemos estado>>8 y mostramos tanto el pid de nuestro hijo como su estado de finalizacion.

Error en el execl: Bad address

Mi hijo 3354 ha finalizado con el estado 65280

En mi caso el programa falla porque no estaba añadido el argumento final de NULL, así que lo he añadido y el resultado ha sido:

linux-vdso.so.1 => (0x00007fff38d64000)

libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f43b798e000)

/lib64/ld-linux-x86-64.so.2 (0x00007f43b7d6e000)

Mi hijo 3445 ha finalizado con el estado 0

Ahora si ha finalizado correctamente e imprime las librerias correctas.