Sistemas Operativos

Formulario de auto-evaluación

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

Nombre y apellidos:

Cristina María Garrido López

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 40 minutos.

1. He resuelto todas las dudas que tenía antes de iniciar la sesión de prácticas: sí. 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:

```
int main(int argc, char *argv[]){
	if(argc!=2){
		printf("Numero de argumentos invalido.");
		exit(-1);
	}
	else{
		int num=atoi(argv[1]);
		pid_t pid;

		if( (pid=fork())<0) {
			perror("\nError en el fork");
			exit(-1);
		}
		else if(pid==0){
			if(num%2==0)
				printf("El numero es par.\n");
			else
				printf("El numero es impar\n.");
		}else{
			if(num%4==0)
				printf("El numero es divisible por 4.\n");
			else
				printf("El numero no es divisible por 4.\n");
		}
	}
}
```

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

En la jerarquía 1, existe un proceso que crea a un hijo y se va, este crea a otro y se va, este

último a otro y así sucesivamente. Para comprobarlo lo he modificado:

```
int main(int argc, char *argv[]){
        pid_t childpid;
        int nprocs=20,i,j,k;
        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: %d\n",i,strerror(errno));
                        exit(-1);
                }
          if (childpid){
                        printf("Jerarq1. Mi PID es %d y el de mi padre %d.\n",getpid(),getppid());
                        break;
                }
        }

        wait();
```

En la jerarquía 2, existe un padre que crea muchos procesos hijo que van saliendo.

```
for (j=0; j < nprocs; j++) {
          if ((childpid= fork()) == -1) {
                fprintf(stderr, "Could not create child %d: %d\n",j,strerror(errno));
                exit(-1);
          }

          if (!childpid){
                printf("Jerarquia2. Mi PID es %d y el de mi padre %d.\n",getpid(),getppid());
            break;
                }
```

```
        }
for(k=0; k<nprocs; k++) wait();
```

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


pid_t hijo;
int main(int argc, char *argv[]){
        pid_t childpid;
        int NUM_HIJOS=5,j,k;
        pid_t vec[NUM_HIJOS];

        if(setvbuf(stdout,NULL,_IONBF,0)) {
                perror("\nError en setvbuf");
        }



        for (j=0; j < NUM_HIJOS; j++) {
                if((vec[j] = fork()) == -1) {
                        fprintf(stderr, "No se pudo crear el hijo %d: %d\n",j,strerror(errno));
                        return -1;
                  }

                if (vec[j]==0){
```

```
                        printf(" Soy el hijo %d\n",getpid());
                        return -1;
                }
        }

        for(k=0; k<NUM_HIJOS; k++){
                waitpid(vec[k]);
                printf("Acaba de finalizar mi hijo con %d\n", vec[k]);
                printf("Solo me quedan %d hijos vivos\n",NUM_HIJOS-k-1);
        }

        return -1;

}
```

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

En este ejercicio se crea un proceso hijo que va a ejecutar el programa ldd, ubicado en /usr/bin/ldd al que se le pasa un único argumento, ./tarea5. Después el proceso padre espera a que este hijo termine e imprime por pantalla su PID y el estado con el que ha finalizado.