Sistemas Operativos

Formulario de auto-evaluación

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

Nombre y apellidos:

Néstor Rodríguez Vico

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 ... 50... minutos.

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

Creación de procesos

3. Comentarios y sugerencias:

b) Cuestionario de conocimientos adquiridos.

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

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
//Usar variables bool
#include <stdbool.h>

bool par(int numero) {
        return numero % 2 == 0;
}

bool divisible(int numero) {
   return numero % 4 == 0;
}

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

   if(argc!=2) {
      printf("Modo de uso: %s <entero>\n\n", argv[0]);
      exit(1);
   }

   numero=atoi(argv[1]);

   if((pid=fork())<0){
      printf("\nError %d al realizar fork",errno);
      perror("\nError en el fork");
      exit(EXIT_FAILURE);
```

```
    }
    //Hijo
    else if(pid==0){
        if (par(numero))
            printf("El numero %d es par\n",numero);
        else
            printf("El numero %d no es par\n",numero);
    }
    //Padre
    else {
        if (divisible(numero))
            printf("El numero %d es divisible por 4\n",numero);
        else
            printf("El numero %d no es divisible por 4\n",numero);
    }
    return EXIT_SUCCESS;
}
```

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

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

int main(){
    pid_t childpid;
    if(setvbuf(stdout, NULL, _IONBF,0)){
        perror("Error en setvbuf. \n");
    }
    // Jerarquía de procesos tipo 1
    for (int i=1; i < 20; i++) {
        if ((childpid= fork()) == -1) {
```

```
            printf("\nError %d al realizar fork",errno);
            exit(-1);
        }
        printf("PID = %d.\t PID del padre = %d\n", getpid(), getppid());
        if (childpid)
            break;
    }

    /*
    // Jerarquía de procesos tipo 2
    for (i=1; i < 20; i++) {
        if ((childpid= fork()) == -1) {
            printf("\nError %d al realizar fork",errno);
            exit(-1);
        }
        printf("PID = %d.\t PID del padre = %d\n", getpid(), getppid());
        if (!childpid)
            break;
    }

    */
    return 0;
}
```

En el primer caso se obtiene una jerarquía con un único padre y muchos hijos.

En el segundo caso se obtiene una jerarquía en la cual un padre crea un hijo, este hijo a otro hijo, este a otro hijo y así sucesivamente.

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

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

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

int main(){
	int i, estado;
	pid_t pid[5];
	int contador=4;
	//CREAMOS HIJOS
	for(i=0; i<5; i++){
		if((pid[i] = fork())<0){
			printf("\nError %d al realizar fork",errno);
			perror("Error en fork\n");
			exit(-1);
		}
		//Hijos
		if(pid[i]==0){
			printf("Soy el hijo PID = %i\n", getpid());
			exit(0);
		}
	}

	for (int i = 0;i<5;i++) {
		if (waitpid(pid[i],0,0) > 0) {
		printf("Acaba de finalizar mi hijo con PID: %d\n",pid[i]);
		printf("Solo me quedan %d hijos vivos\n", contador);
			contador = contador - 1;
	}
	}
	return 0;
}
```

Un ejemplo de ejecución es:

```
Soy el hijo PID = 7874
Soy el hijo PID = 7877
Soy el hijo PID = 7878
Soy el hijo PID = 7875
Soy el hijo PID = 7876
Acaba de finalizar mi hijo con PID: 7874
Solo me quedan 4 hijos vivos
Acaba de finalizar mi hijo con PID: 7875
Solo me quedan 3 hijos vivos
Acaba de finalizar mi hijo con PID: 7876
Solo me quedan 2 hijos vivos
Acaba de finalizar mi hijo con PID: 7877
Solo me quedan 1 hijos vivos
Acaba de finalizar mi hijo con PID: 7878
Solo me quedan 0 hijos vivos
```

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

Se crea un proceso hijo y este ejecuta el programa /usr/bin/ldd con los argumentos “ldd” y “./tarea5”. Esto hace que se muestren las dependencias de bibliotecas compartidas del archivo tarea5 (el archivo ejecutable de la tarea 5).