Sistemas Operativos

Formulario de auto-evaluación

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

Nombre y apellidos:

Juan Carlos Ruiz García

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:

En clase resolví todas las dudas durante la explicación de la práctica.

2. Tengo que trabajar algo más los conceptos sobre:

Por el momento lo llevo todo bien.

3. Comentarios y sugerencias:

Ningun comentario.

b) Cuestionario de conocimientos adquiridos.

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

Adjunto llamado ejer1.c (De todas formas se lo pongo a continuación en texto).

```
/*
Nombre: Juan Carlos Ruiz García
Grupo: C2
Año: 2016
*/

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

```c
int main(int argc, char *argv[])
{
pid_t pid;
int num;

if(argc != 2){
printf("Numero de argumentos incorrecto\nSintaxis:\t\t\tEjemplo:\n");
printf("%s <numero_entero>\t\t%s 5\n",argv[0],argv[0]);
exit(EXIT_FAILURE);
}

num = atoi(argv[1]);

if( (pid=fork())<0) {
perror("\nError en el fork");
exit(EXIT_FAILURE);
}
else if(pid==0) { //proceso hijo comprobando si el numero es par o impar
if( num%2 == 0 )
printf("El numero %d es un numero par.\n",num);
else
printf("El numero %d es un numero impar\n",num);
exit(1);
}

if( num%4 == 0 )
printf("El numero %d es divisible por 4.\n",num);
else
printf("El numero %d no es divisible por 4.\n",num);

printf("PID Padre: %d\nPID Hijo: %d\n",getpid(),pid);

exit(EXIT_SUCCESS);

}
```

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

Adjuntos llamados ejer3_a.c y ejer3_b.c (De todas formas se lo pongo a continuación en texto).

```c
/*
Nombre: Juan Carlos Ruiz García
Grupo: C2
Año: 2016
Fichero: ejer3_a.c
*/

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

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

  pid_t childpid;
  int nprocs = 20;
```

```
  /*
  Jerarquía de procesos tipo 1

  Este tipo de jerarquia generaria nprocs hijos de un nuevo padre en cada iteracion

  */
  for (int i=1; i < nprocs; i++) {
    if( (childpid= fork()) == -1 ) {
        fprintf(stderr, "Could not create child %d: %s\n",i,strerror(errno));
        exit(-1);
    }
    // printf("%d\n",childpid );
    if (childpid)
      break;
    printf("%d.Mi PID: %d\t\tPID padre: %d\n",i,childpid,getppid());
  }

  exit(EXIT_SUCCESS);

}

/*
Nombre: Juan Carlos Ruiz García
Grupo: C2
Año: 2016
Fichero: ejer3_b.c
*/

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

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

 pid_t childpid;
 int nprocs = 20;

 /*
 Jerarquía de procesos tipo 1

 Este tipo de jerarquia generaria nprocs hermanos del mismo padre

 */
 for (int 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)
   break;
  printf("%d.Mi PID: %d\t\tPID padre: %d\n",i,childpid,getppid());
 }

 exit(EXIT_SUCCESS);

}
```

Ejemplo de ejecucion de ambos tipos de jerarquia.

```
juanka1995@juanka-laptop ~/practicas/so/MOD2-Sesion3/src $ ./ejer3_a
1.Mi PID: 0             PID padre: 15358
juanka1995@juanka-laptop ~/practicas/so/MOD2-Sesion3/src $ 2.Mi PID: 0          PID padre: 1
3.Mi PID: 0             PID padre: 15360
4.Mi PID: 0             PID padre: 15361
5.Mi PID: 0             PID padre: 15362
6.Mi PID: 0             PID padre: 1
7.Mi PID: 0             PID padre: 15364
8.Mi PID: 0             PID padre: 15365
9.Mi PID: 0             PID padre: 15366
10.Mi PID: 0            PID padre: 15367
11.Mi PID: 0            PID padre: 15368
12.Mi PID: 0            PID padre: 15369
13.Mi PID: 0            PID padre: 15370
14.Mi PID: 0            PID padre: 15371
15.Mi PID: 0            PID padre: 15372
16.Mi PID: 0            PID padre: 15373
17.Mi PID: 0            PID padre: 1
18.Mi PID: 0            PID padre: 15375
19.Mi PID: 0            PID padre: 15376

juanka1995@juanka-laptop ~/practicas/so/MOD2-Sesion3/src $ ./ejer3_b
1.Mi PID: 15379         PID padre: 3922
2.Mi PID: 15380         PID padre: 3922
3.Mi PID: 15381         PID padre: 3922
4.Mi PID: 15382         PID padre: 3922
5.Mi PID: 15383         PID padre: 3922
6.Mi PID: 15384         PID padre: 3922
7.Mi PID: 15385         PID padre: 3922
8.Mi PID: 15386         PID padre: 3922
9.Mi PID: 15387         PID padre: 3922
10.Mi PID: 15388                PID padre: 3922
11.Mi PID: 15389                PID padre: 3922
12.Mi PID: 15390                PID padre: 3922
13.Mi PID: 15391                PID padre: 3922
14.Mi PID: 15392                PID padre: 3922
15.Mi PID: 15393                PID padre: 3922
16.Mi PID: 15394                PID padre: 3922
17.Mi PID: 15395                PID padre: 3922
18.Mi PID: 15396                PID padre: 3922
19.Mi PID: 15397                PID padre: 3922
juanka1995@juanka-laptop ~/practicas/so/MOD2-Sesion3/src $
```

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

Adjunto llamado ejer4.c (De todas formas se lo pongo a continuación en texto)

```c
/*
Nombre: Juan Carlos Ruiz García
Grupo: C2
Año: 2016
*/

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

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

 pid_t pid;
 int nprocs = 5, estado;

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

```
  if( (pid = fork()) < 0 ) {
    fprintf(stderr, "No se pudo crear el hijo %d: %s\n",i,strerror(errno));
    exit(EXIT_FAILURE);
  }
  if(pid == 0){
    printf("Soy el hijo %d\n",getpid());
    exit(EXIT_SUCCESS);
  }
 }
 for (int i = 0; i < nprocs; i++) {
  pid = wait(&estado);
  printf("Acaba de finalizar mi hijo con PID %d\n",pid);
  printf("Solo me quedan %d hijos vivos\n",nprocs-i-1);
 }

 exit(EXIT_SUCCESS);

}
```

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

En este programa lo que ocurre es que se crea una hebra hija de la hebra padre creada al lanzar el programa. Esta hebra realiza una llamada exec al ejecutable ldd, en mi caso lo he hecho con el ejecutable ls ya que es algo mas comun de usar.

En este momento el espacio de direcciones de usuario de la hebra hija se reemplaza totalmente por el espacio de direcciones del ejecutable ls que se le paso como argumento a la orden exec.

Termina la ejecucion del ls y la hebra termina. Posteriormente la hebra padre muestra el mensaje correspondiente por pantalla.

```
juanka1995@juanka-laptop ~/practicas/so/MOD2-Sesion3/src $ ./tarea5
5644241 -rwxr-xr-x 1 juanka1995 juanka1995 8856 Nov 19 19:35 ./tarea5

Mi hijo 17553 ha finalizado con el estado 0
```

```
/*
tarea5.c
Trabajo con llamadas al sistema del Subsistema de Procesos conforme a POSIX 2.10
*/

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

int main(int argc, char *argv[]){
      pid_t pid;
      int estado;
      if( (pid=fork())<0) {
            perror("\nError en el fork");
            exit(-1);
      }
      else if(pid==0) { //proceso hijo ejecutando el programa
            if( (execl("/bin/ls","ls","-li","./tarea5",NULL)<0)) {
```

```
                perror("\nError en el execl");
                exit(-1);
            }
        }
        wait(&estado);
        printf("\nMi hijo %d ha finalizado con el estado %d\n",pid,estado>>8);

        exit(0);

}
```