Sistemas Operativos

Formulario de auto-evaluación

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

Nombre y apellidos:

Antonio Miguel Morillo Chica

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 10 minutos.

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

Ninguna.

2. Tengo que trabajar algo más los conceptos sobre:

Como crear procesos con los bucles. Realmente me ha costado un poco esta práctica. Sobretodo por el uso del wait y waitpid.

3. Comentarios y sugerencias:

Ahondar en clase de teoría en el uso del wait y waitpid con bucles.

b) Cuestionario de conocimientos adquiridos.

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

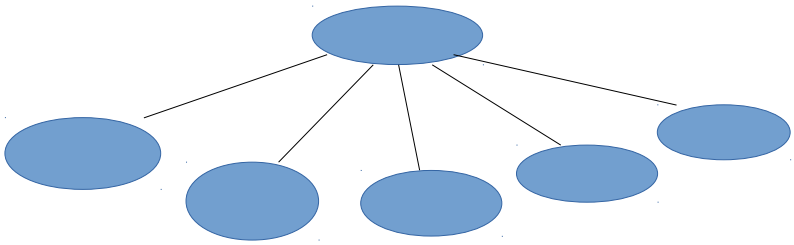
```
#include<sys/types.h>
#include<unistd.h>
#include<stdio.h>
#include<errno.h>
#include <stdlib.h>

void comprobar_hijo(int valor){
  if (valor % 2 == 0)
    printf("Hijo par\n");
  else
    printf("Hijo impar\n");
}

void comprobar_padre(int valor){
  if (valor % 4 == 0)
    printf("Padre par\n");
  else
    printf("Padre impar\n");
}

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

  if(argc < 1){
    printf("Sintaxis de ejecucion: ejercicio1 [<numero_int>]\n");
    exit(-1);
  }

  int n = atoi(argv[1]);
  pid_t pid;

  if( (pid=fork())<0) {
          perror("\nError en el fork");
          exit(-1);
  }

  // Si es un hijo lo compruebo
  if (pid==0) {
    comprobar_hijo(n);
  } else { // es un padre
    comprobar_padre(n);
  }

  return 0;
}
```

Mi solución a la **ejercicio 3** 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 argc, char const *argv[]) {
   int estado;
   int nprocs = atoi(argv[1]);
   pid_t childpid;

   if (argc != 2) {
      printf("Sintaxis de ejecucion: ejercicio1 [<numero_int>]\n");
      exit(-1);
   }

   for(int i=0; i<nprocs; ++i){
      if ((childpid = fork()) == -1) {
         fprintf(stderr, "Could not create child %d: s\n", i, strerror(errno));
         exit(-1);
      }
      // Espero a que acabe el hijo
      wait(childpid);

      //Slgo del buble con el hijo
      if(childpid)          break;
   }

   // Escribire toda la info de los hijos y por último el del padre
   if(childpid) printf("PID: %d \tPID padre: %d\n",getpid(),getppid());
   else printf("PID: %d \tPID padre: %d\n",getpid(),getppid());

 /*    Es el mismo código pero ordena la infor en distinto orden.

   for(int i=0; i<nprocs; ++i){
      if ((childpid = fork()) == -1)
         fprintf(stderr, "Could not create child %d: s\n", i, strerror(errno));
         exit(-1);
      }
      wait(&estado);
      if(!childpid)    break;
 }

    if(childpid)     printf("PID: %d \tPID padre: %d\n",getpid(),getppid());
    else             printf("PID: %d \tPID padre: %d\n",getpid(),getppid());
*/

  return 0;
}
```

PID: 14479 PID padre: 14478	PID: 13750 PID padre: 13749
PID: 14478 PID padre: 14477	PID: 13751 PID padre: 13749
PID: 14477 PID padre: 14476	PID: 13752 PID padre: 13749
PID: 14476 PID padre: 14475	PID: 13753 PID padre: 13749
PID: 14475 PID padre: 14474	PID: 13754 PID padre: 13749
PID: 14474 PID padre: 14473	PID: 13755 PID padre: 13749
PID: 14473 PID padre: 14472	PID: 13756 PID padre: 13749
PID: 14472 PID padre: 14471	PID: 13757 PID padre: 13749
PID: 14471 PID padre: 14470	PID: 13758 PID padre: 13749
PID: 14470 PID padre: 14469	PID: 13759 PID padre: 13749
PID: 14469 PID padre: 14468	PID: 13760 PID padre: 13749
PID: 14468 PID padre: 14467	PID: 13761 PID padre: 13749
PID: 14467 PID padre: 14466	PID: 13762 PID padre: 13749
PID: 14466 PID padre: 14465	PID: 13763 PID padre: 13749
PID: 14465 PID padre: 14464	PID: 13764 PID padre: 13749
PID: 14464 PID padre: 14463	PID: 13765 PID padre: 13749
PID: 14463 PID padre: 14462	PID: 13766 PID padre: 13749
PID: 14462 PID padre: 14461	PID: 13767 PID padre: 13749
PID: 14461 PID padre: 14460	PID: 13768 PID padre: 13749
PID: 14460 PID padre: 14459	PID: 13769 PID padre: 13749
PID: 14459 PID padre: 14422	PID: 13749 PID padre: 11666

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 vivos = 5;

int main(int argc, char const *argv[]) {
  int estado;
  int nprocs = 5;
  pid_t childpid;

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

```
   // Hago que el padre espere al hijo
   waitpid(-1,&estado,1);

   // Si es un hijo rompo el bucle en caso contrario
   // lo que hago es escribir que ya ha acabado el hijo
   if(childpid)      break;
   else{
      vivos=vivos-1;
      printf("Acaba de finalizar mi hijo con %d\nSolo me quedan %d hijos vivos.\n\n",getpid(),vivos);
    }
  }

   // Hago que el hijo escriba su pid y termino el proceso (exit)
   // en caso contrario espero a que finalicen todas los procesos hijo
   if(childpid){
      printf("Soy el hijo: %d\n",getpid());
      exit(0);
   }else     waitpid(-1,&estado,1);

  return 0;
}
```

Mi solución a la **ejercicio 6** 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 argc, char *argv[]){
  pid_t pid;
  int estado;

  if( (pid=fork())<0) { perror("\nError en el fork"); exit(-1);
  }else if(pid==0) {
      /*
        Si es un hijo el programa ejecuta ldd con la opcion execl que necesita la ruta
        del ejecutable, el nombre del ejecutable y todos los argumentos. Hay que
        destacar que el segundo argumentos de la función es el argv[0].
      */
          if( (execl("/usr/bin/ldd","ldd","./tarea5")<0)) {
                  perror("\nError en el execl");
                  exit(-1);
          }
   }
  // El padre espera, el hijo hará esto antes y despues el padre.
  wait(&estado);
  printf("\nMi hijo %d ha finalizado con el estado %d\n",pid,estado>>8);
  // Valor de retorno para el wait(&estado) del padre
  exit(0);
}
```