Sistemas Operativos

Formulario de auto-evaluación

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

Nombre y apellidos:

David Sánchez Jiménez

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

void comprobar_paridad(int value) {
  if (value % 2 == 0) {
    printf("Hola, soy el hijo y el numero es par. \n");
  } else {
    printf("Hola, soy el hijo y el numero es impar. \n");
  }
}

void comprobar_divisibilidad(int value) {
  if (value % 4 == 0) {
    printf("Hola, soy el padre y el numero es divisible entre 4. \n");
  } else {
    printf("Hola, soy el padre y el numero no es divisible entre 4. \n");
  }
}

int main(int argc, char const *argv[]) {
  if (argc != 2) {
    perror("\nError: ./ejercicio1 <numero_comprobar>");
    exit(-1);
  } else {
    pid_t pid = fork();

    if (pid == 0) {
      comprobar_paridad(atoi(argv[1]));
    } else if (pid > 0) {
      comprobar_divisibilidad(atoi(argv[1]));
    }
  }

  return 0;
}
```

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

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

void main(int argc, char *argv[]) {
  int nprocs = 20;
  int i;
  pid_t childpid;

  // Jerarquia de procesos tipo 1
  // Cada hijo genera un su hijo

  printf("Jerarquia de procesos Tipo 1:\n");

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

    sleep(3);

    printf("childpid = %d, parentpid = %d \n", getpid(), getppid());

    if (childpid) {
      break;
    }
  }

  // Jerarquia de procesos tipo 2
  // Solo el padre inicial puede generar hijos

  printf("Jerarquia de procesos Tipo 2:\n");

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

    sleep(3);

    printf("childpid = %d, parentpid = %d \n", getpid(), getppid());

    if (!childpid) {
      break;
    }
  }
}
```

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

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

int main(int argc, char const *argv[]) {
  int i;
  const int NUM_HIJOS = 5;
  pid_t pid[NUM_HIJOS];

  for (int i = 0; i < NUM_HIJOS; i++) {
    if ((pid[i] = fork()) == 0) {
      printf("Soy el hijo PID %d \n", getpid());
      break;
    } else if (pid[i] < 0) {
      printf("Error in creating child process \n.");
      return -1;
    }
  }

  for (int i = 0; i < NUM_HIJOS; 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", NUM_HIJOS - (i + 1));
    }
  }

  return 0;
}
```

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

El código del archivo tarea5.c genera un proceso hijo el cual con el comando ldd va mostrando por pantalla las bibliotecas compartidas que necesita para ejecutarse. Una vez se ejecuta la orden ldd y el hijo ha terminado (estado = 0) o se ha generado un error (estado != 0) el padre envía un mensaje con el PID del hijo acabado y su estado.