Presentación

Esta práctica plantea una serie de actividades con el objetivo que el estudiante pueda aplicar sobre un sistema UNIX algunos de los conceptos introducidos en los primeros módulos de la asignatura.

El estudiante deberá realizar una serie de experimentos y responder a las preguntas planteadas. También deberá escribir un programa en lenguaje C.

Competencias

Transversales:

- Capacidad para adaptarse a las tecnologías y a los futuros entornos actualizando las competencias profesionales

Específicas:

- Capacidad de analizar un problema en el nivel de abstracción adecuado a cada situación y aplicar las habilidades y conocimientos adquiridos para abordarlo y resolverlo

- Capacidad de diseñar y construir aplicaciones informáticas mediante técnicas de desarrollo, integración y reutilización

Enunciado

Para realizar esta práctica os facilitamos el fichero **pr2so.zip** con ficheros fuente. Descomprimidlo con la orden **unzip pr2so.zip**. Para compilar un programa, por ejemplo, **prog.c**, debéis ejecutar la orden **gcc -o prog prog.c**

1. (4 puntos = 1+1+2) Dados los siguientes códigos, indicad cuál es el resultado de ejecutarlos y justificar adecuadamente la respuesta a cada pregunta.

   Se aconseja que en primer término intentéis determinar su comportamiento sin ejecutarlos en terminal. Ejecutadlos para verificar vuestra respuesta. Podéis asumir que ninguna llamada devolverá error.

   No tenéis que *corregir* los códigos. Sólo se pide que expliquéis su comportamiento.

a) Código 1 (1 punto): Explica qué hace cada uno de los dos códigos, y después de ejecutarlos varias veces, indica cuantos "Hola" muestra cada uno por pantalla. ¿Es el resultado que esperabas?

```c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
        fork();
        printf("Hola ");

    return 0;
}
```

a) fork() antes del printf()

```c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
        printf("Hola ");
        fork();

    return 0;
}
```

b) fork() después del printf()

Sol: El código (a) crea un proceso hijo, y después tanto padre como hijo ejecutar la función printf mostrando por el terminal un "Hola" cada uno. El código (b), el proceso principal ejecuta la función printf y luego crea un hijo, al no haber más funciones los dos procesos terminan. En el caso (b) también se muestran dos "Hola". Intuitivamente se esperaba un único "Hola" el en caso b ya que el fork está después de la ejecución de la función printf.

b) Código 2 (1 punto): A partir del siguiente código, ¿qué proceso ejecuta cada printf?

```c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
        fork();
        printf("Hola soy %d", getpid());

    return 0;
}
```

a) fork() antes del printf()

```c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
        printf("Hola soy %d", getpid(),);
        fork();

    return 0;
}
```

b) fork() después del printf()

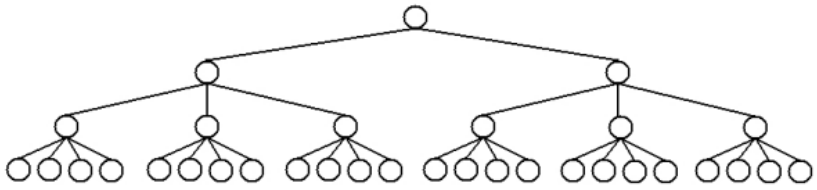
Sol: En el código (b) el printf es ejecutado únicamente por el proceso padre, esto lo podemos deducir porque muestra el mismo pid en el terminal "Hola soy 112115 Hola soy 112115". En el código (a) tanto el padre como el hijo ejecutan la función printf, "Hola soy 112115 Hola soy 112116".

c) (2 puntos): ¿Cómo es posible que en los códigos (a) anteriores, la mayoría de las ejecuciones se muestre por pantalla dos veces el texto contenido en printf? ¿El proceso hijo ejecuta el printf de los códigos (a) en algún caso?

Sol: Cuando se envía a la salida estándar usando la printf() función de la biblioteca C, la salida generalmente se almacena en búfer. El búfer no se vacía hasta que genera una nueva línea, llama fflush(stdout) o sale del programa (_exit()aunque no a través de la llamada). Cuando el proceso principal realiza el fork(), el proceso hijo hereda cada parte del proceso principal, incluido el búfer de salida sin vaciar. Esto copia efectivamente el búfer no vaciado en cada proceso secundario. Cuando termina el proceso, los búferes se vacían. Inicia un gran total de ocho procesos (incluido el proceso original) y el búfer no vaciado se vaciará al final de cada proceso individual. En ningún caso el hijo ejecuta el printf.

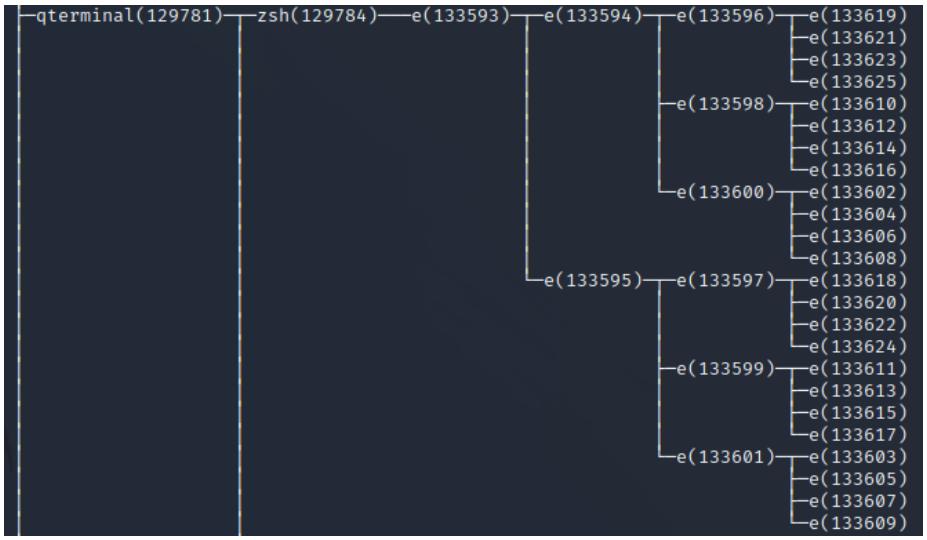
2. A continuación, se pide responder a las siguientes cuestiones de la forma más clara posible. (6 puntos = 3 +3)

2.1. (3 puntos) Completar el código **arbol.c** proporcionado el zip para que genere un árbol de procesos similar a la siguiente figura. Escribe el fragmento de código que genera la jerarquía de procesos que se muestra en la figura para profundidad n. Observa que en la figura siguiente la profundidad n=3.



```
#include <stdio.h>
#include <unistd.h>

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

        int i, j, nuevo, pid, npid, n;

        n=3;
        for (i= 0; i</* tu código aquí */; i++){
          pid= getpid();

          for(j=0;j</* tu código aquí */; j++){
                    nuevo= fork();

                    if (/* tu código aquí */) break;

          }
          npid= getpid();
          if (/* tu código aquí */) break;
        }

        sleep(60); /* permite comprobar con pstree -p la generación del árbol de procesos */

   return 0;
}
```

Nota: Para comprobar si la generación del árbol de procesos se ha realizado correctamente, añadimos un *sleep* de 60 segundos para que nos dé tiempo a abrir otro terminal y utilizar *pstree -p*.



Sol:

```
#include <stdio.h>
#include <unistd.h>

int main() {

        int i, j, nuevo, pid, npid, n;

        n=3;
        for (i= 0; i<n; i++){
          pid= getpid();

          for(j=0;j<i+2; j++){
                  nuevo= fork();

                  if (nuevo == 0) break;

          }
          npid= getpid();
          if (npid== pid) break;
        }

        sleep(60);

  return 0;
}
```

2.2 (3 puntos)

Escribid un programa en C que pasándole dos cadenas de caracteres como argumento después escriba el número de caracteres de cada cadena por pantalla. El conteo de la primera cadena lo deberá realizar el proceso hijo, y el padre el de la segunda cadena. El proceso padre deberá esperar a que termine el proceso hijo antes de mostrar por pantalla el número de parámetros de su cadena.

```
└─$ ./ejercicio2 hola adios
Numero caracteres Parametro 1= 4
Numero caracteres Parametro 2= 5
```

Sol:

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

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

    pid = fork();
    if (pid == 0)
    {
            int nc=0;
        for (pos = 0; argv[1][pos] != '\0'; pos++)
            nc++;

        printf("Numero caracteres Parametro 1= %d\n",nc);

    }else{
            int nc=0;
        for (pos = 0; argv[2][pos] != '\0'; pos++)
            nc++;
        wait(NULL);
        printf("Numero caracteres Parametro 2= %d\n",nc);

    }
    return 0;
}
```

Recursos

- Módulos 1, 2, 3, 4, 5 y 6 de la asignatura.
- Documento ”Introducción a la programación de Unix” (disponible en el aula) o cualquier otro manual similar.
- El aula ”Laboratorio de Sistemas Operativos” (podéis plantear vuestras dudas relativas al entorno UNIX, programación,...).

Criterios de evaluación

Se valorará la justificación de las respuestas presentadas.

El peso de cada pregunta esta indicado en el enunciado.

Formato y fecha de entrega

Se creará un fichero **zip** que contendrá todos los ficheros de la entrega (un **pdf** con la respuesta a las preguntas y ficheros fuente).

Fecha límite de entrega: 22/12/2022