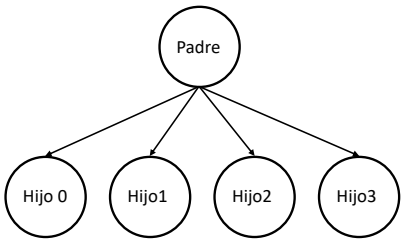
1. Realice un programa que usando procesos pesados `fork` cree 4 procesos hijos, tal que todos sean hermanos entre sí tal como se muestra en la figura, es decir, que todos sean hijos directos del programa principal `main()`. El programa padre deberá esperar a que todos los hijos terminen para poder terminar.



   Cada hijo escribirá un mensaje en la salida estándar indicando que es un proceso hijo, su número de hijo, cuál es su `PID` y cuál es el `PID` del padre (se puede usar para esto las funciones `getpid()` y `getppid()`). Si la creación de hijos es correcta el PID del padre debe ser el mismo para todos.

2. Modifique el ejercicio 1 para que cada hijo haga lo siguiente: escribirá en orden creciente 100 números, uno en cada línea y con tres cifras. Los números estarán *encolumnados* a partir de la columna n, siendo n el número de hijo, usando como separador de columnas el tabulador. El primer número de cada hijo (para diferenciar la secuencia) será el (100*n). El código a ejecutar por os hijos ha de ser el mismo en todos los casos. Un ejemplo de salida podría ser la siguiente:

```
000
001
002
      101
      102
      103
...
            200
            201
            202
            203
003
004
...

                  300
                  301
                  302
                  303
      104
      105
...
```

   Ejecute el proceso varias veces. La salida no debe ser la misma ¿Por qué?

Los siguientes ejercicios permiten ver ejemplos del uso de funciones de la familia `exec` y las funciones `fork` y `exec`. Codifique y entienda el comportamiento de cada uno de ellos.

3. Uso de la función `execl`:

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

int main() {
  char *path = "/bin/ls";
  char *comando = "ls";
  char *arg1 ="-l";
  char *arg2 ="/home";
  int estado;

  printf("Antes de llamar a execl()\n");
  if ((estado=execl(path, comando, arg1, arg2, NULL)) == -1) {
    printf("El proceso no ha terminado correctamente\n");
    exit(1);
  }

  printf("Esta linea NO se saldra por la salida estandar si la llamada a
   execvp() termina correctamente\n");

  return 0;
}
```

4. Uso de la función `execl` y `fork`:

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

int main() {
  char *path = "/bin/ls";
  char *comando = "ls";
  char *arg1 ="-l", *arg2 = "/home";
  int estado;

  printf("Antes de llamar a execl()\n");
  printf("\tCreamos otro proceso usando fork()...\n");
  if (fork() == 0) {
    int estado = execl(path, comando, arg1, arg2, NULL);
    printf("\t\tls -l /home  ha tomado el control de este proceso hijo.
   Esto no se ejecutara a menos que termine de manera anormal!\n");
    if (estado == -1) {
      printf("\t\tEl proceso no ha terminado correctamente\n");
      return 1;
    }
  }
  else {
    printf("Esta linea se ejecutara por el proceso padre\n");
    wait(NULL);
  }
  return 0;
}
```

5. Uso de la función `execle`:

```c
#include <unistd.h>

int main(void) {
  char *path = "/bin/sh";
  char *comando = "sh";
  char *arg1 = "-c";
  char *arg2 = "echo $0 $SHELL";
  char *env[] = {"SHELL=Hola", NULL};

  execle(path, comando,arg1, arg2, NULL, env);

  return 0;
}
```

6. Uso de la función `execvp`:

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

int main() {
  char *comando = "ls";
  char *lista_argumentos[] = {"ls", "-l", NULL};
  int estado;

  printf("Antes de llamar a execvp()\n");
  if (( estado  = execvp(comando, lista_argumentos)) == -1) {
    printf("El proceso no ha terminado correctamente\n");
    exit(1);
  }
  printf("Esta linea NO se saldra por la salida estandar si la llamada a
   execvp() termina correctamente\n");
  return 0;
}
```

7. Uso de la función `execvp` y `fork`:

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

int main() {
  char *comando = "ls";
  char *lista_argumentos[] = {"ls", "-l", NULL};
  int estado;

  printf("Antes de llamar a execvp()\n");
  printf("\tCreamos otro proceso usando fork()...\n");
  if (fork() == 0) {
    int estado = execvp(comando, lista_argumentos);
    printf("\t\tls -l ha tomado el control de este proceso hijo. Esto no
   se ejecutara a menos que termine de manera anormal!\n");
    if (estado == -1) {
      printf("\t\tEl proceso no ha terminado correctamente\n");
      return 1;
    }
```

```
19    }
20    else {
21      printf("Esta linea se ejecutara por el proceso padre\n");
22      wait(NULL);
23    }
24    return 0;
25  }
```