Práctica 2 - Procesos en C

Ejercicio 1

Crea un programa llamado ejemploFork.c en el que crees un proceso con fork() y el padre imprima por pantalla "Hola, soy el proceso padre con PID:%d y PPID:%d.\nHe creado un proceso con PID %d ", mientras que el hijo imprima por pantalla "Hola, soy el proceso hijo con PID:%d y PPID: %d".
Recuerda utilizar el wait para no generar procesos zombies y controlar el posible error de fork()

cfgs@LAB-35-PC13: ~/Procesos

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

void main(){
        pid_t pid, Hijo_pid;
        pid = fork();

        if (pid == -1)
        {
                printf("No se ha podido crear el proceso hijo...");
                exit(-1);
        }
        if (pid == 0) // Nos encontramos en el proceso hijo
        {

                printf("Hola soy el proceso hijo con PID: %d y PPID: %d.\n",getpid(),getppid());
        }
        else // Nos encontramos en el proceso padre

                //3er paso
                //Hijo_pid = wait(NULL); //Espera la finalización del proceso hijo.
                printf("Hola soy el proceso padre con PID %d y PPID: %d.\n\t. He creado un proceso con PID: %d .\n",getpid(),getppid(),pid);
        }
```

```
cfgs@LAB-35-PC13:~/Procesos$ gcc -o ejemploFork ejemploFork.c
cfgs@LAB-35-PC13:~/Procesos$ ./ejemploFork
Hola soy el proceso padre con PID 11290 y PPID: 350.
        . He creado un proceso con PID: 11291 .
Hola soy el proceso hijo con PID: 11291 y PPID: 11290.
cfgs@LAB-35-PC13:~/Procesos$
```

¿Coincide algún PID o algún PPID? ¿Por qué?
Si coinciden el PID del padre es igual el PPID del hijo porque este se inicia con fork 0 por lo cual el PID es distinto pero se realiza una vez empieza el padre.

Ejercicio 2

Crea un programa en el que se cree la siguiente jerarquía de procesos:
Un padre tiene 2 hijos, H1 y H2. H2, a su vez, tiene 2 hijos. Cada uno deberá terminar con un entero que indique su nivel en la jerarquía (El padre 0, los hijos del padre 1, los nietos 2)
Lo único que tiene que hacer cada proceso es imprimir por pantalla un mensaje con la siguiente información:

- PID del proceso
- PPID del proceso
- PID del proceso hijo que ha creado (si es que ha creado uno).
- Código de retorno de los procesos creados.

Recuerda utilizar el wait para no generar procesos zombies y controlar el posible error de fork()

cfgs@LAB-35-PC13: ~/Procesos

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

void create_chil(int level, int child_num){
        pid_t pid=fork();

        if(pid < 0){
                //no creado
                perror("No se puedo crear");
                exit(1);
        }else if (pid ==0){
                //hijo1
                printf("Nivel: %d, PID: %d, PPID: %d\n", level,getpid(),getppid());

                //creamos nieto
                if (level ==1){
                        for (int i=1; i<=2;i++){
                                create_chil(2,i);
                        }
                }//salimos del proceso hijo
                 exit (0);
        }else{
                //proceso padre
                int status;
                waitpid(pid,&status,0);//espera al hijo
        }
}

int main(){
        printf("Padre PID: %d\n",getpid());

        //hijo1
        create_chil(1,1);

        //hijo2
        create_chil(1,2);
        return 0;
}
```

```
cfgs@LAB-35-PC13:~/Procesos$ gcc -o jerarqiaDeProcesos jerarquiaDeProcesos.c
cfgs@LAB-35-PC13:~/Procesos$ ./jerarqiaDeProcesos
Padre PID: 12405
Nivel: 1, PID: 12406, PPID: 12405
Nivel: 2, PID: 12407, PPID: 12406
Nivel: 2, PID: 12408, PPID: 12406
Nivel: 1, PID: 12409, PPID: 12405
Nivel: 2, PID: 12410, PPID: 12409
Nivel: 2, PID: 12411, PPID: 12409
cfgs@LAB-35-PC13:~/Procesos$
```

Ejercicio 3:

Parte 1

Crea un programa que calcule la suma de los números primos entre 1 y 1000000 y, después, calcule la suma de los números impares entre 1 y 5000000.
Puedes utilizar unsigned long long int como tipo de dato

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

bool es_primo(int num) {
            if (num < 2) return false;
                for (int i = 2; i * i <= num; i++) {
                                if (num % i == 0) return false;
                                    }
                    return true;
}

int main() {
            unsigned long long int suma_primos = 0;
                unsigned long long int suma_impares = 0;

                    // Calcular la suma de números primos entre 1 y 10,000,000
                         for (int i = 1; i <= 10000000; i++) {
                                 if (es_primo(i)) {
                                                 suma_primos += i;
                                                         }
                                                             }

                                                                                 // Calcular la suma de números impares entre 1 y 5,000,000
                                                                                             for (int i = 1; i <= 5000000; i += 2) {
                                                                                                     suma_impares += i;
                                                                                                         }

                                                                                                                 printf("Suma de primos entre 1 y 10,000,000: %llu\n", suma_primos);
                                                                                                                 printf("Suma de impares entre 1 y 5,000,000: %llu\n", suma_impares);

                                                                                                                     return 0;
}
```

```
cfgs@LAB-35-PC13:~/Procesos$ gcc -o calculoSecuencial calculoSecuencial.c
cfgs@LAB-35-PC13:~/Procesos$ ./calculoSecuencial
Suma de primos entre 1 y 10,000,000: 3203324994356
Suma de impares entre 1 y 5,000,000: 6250000000000
```

Parte 2

Crea un programa que utilice la programación concurrente de procesos para computar en paralelo dos cálculos complejos.
El primer subproceso debe calcular la suma de los números primos entre 1 y 1,000,000.
El segundo subproceso debe calcular el producto de los números impares entre 1 y 500,000 (por simplicidad, puedes limitar el resultado en caso de desbordamiento utilizando el tipo de dato unsigned long long int).
Recuerda hacer wait tantas veces como forks hayas hecho

cfgs@LAB-35-PC13: ~/Procesos

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

//Funcion para verificar si es primo
bool es_primo(int num){
        if(num<2)
                return false;
        for(int i=2;i*i<=num;i++){
                if(num % i==0)
                        return false;
        }

        return true;
}

//Funcion para calcular la suma
void suma_primos(){
unsigned long long suma_primos(int num){
        if (num < 2)
                return false;
        for (int i = 2; i * i <= num; i++) {
                if (num % i == 0)
                        return false;
        }
        return true;
}
}

//Funcion para calcular suma impares
void productos_impares(){
unsigned long long int producto = 1;
for (int i = 1; i <= 500000; i += 2) {
        producto *= i;
        if (producto > 10000000000000000) { // Evitar desbordamiento
                printf("Producto de impares limitado debido a tamaño: %llu\n", producto);
                return;
        }
        printf("Producto de impares entre 1 y 500,000: %llu\n", producto);

}
}

int main(){
        pid_t pid1, pid2;

        // Crear el primer proceso para sumar primos
        pid1 = fork();
        if (pid1 == 0) {
                // Proceso hijo 1 - suma de primos
                suma_primos();
                exit(0);
        //Crear segundo proceso
        pid2=fork();
        if (pid2==0){
                //Proceso hijo 2-producto impares
                productos_impares();
                exit(0);
        }
        //Proceso padre espera a los hijos
```

cfgs@LAB-35-PC13: ~/Procesos

```c
                if(num % i==0)
                        return false;
        }

        return true;
}

//Funcion para calcular la suma
void suma_primos(){
unsigned long long suma_primos(int num){
        if (num < 2)
                return false;
        for (int i = 2; i * i <= num; i++) {
                if (num % i == 0)
                        return false;
        }
        return true;
}
}

//Funcion para calcular suma impares
void productos_impares(){
unsigned long long int producto = 1;
for (int i = 1; i <= 500000; i += 2) {
        producto *= i;
        if (producto > 10000000000000000) { // Evitar desbordamiento
                printf("Producto de impares limitado debido a tamaño: %llu\n", producto);
                return;
        }
        printf("Producto de impares entre 1 y 500,000: %llu\n", producto);

}
}

int main(){
        pid_t pid1, pid2;

        // Crear el primer proceso para sumar primos
        pid1 = fork();
        if (pid1 == 0) {
                // Proceso hijo 1 - suma de primos
                suma_primos();
                exit(0);
        //Crear segundo proceso
        pid2=fork();
        if (pid2==0){
                //Proceso hijo 2-producto impares
                productos_impares();
                exit(0);
        }
        //Proceso padre espera a los hijos
        wait(NULL);
        wait(NULL);

        printf("Calculos completadps en paralelo. \n");
        return 0;
}
```

```
cfgs@LAB-35-PC13:~/Procesos$ vim calculoCocurrente1.c
cfgs@LAB-35-PC13:~/Procesos$ gcc -o calculoConcurrente1 calculoCocurrente1.c
cfgs@LAB-35-PC13:~/Procesos$ ./calculoConcurrente1
Producto de impares entre 1 y 500,000: 1
Producto de impares entre 1 y 500,000: 3
Producto de impares entre 1 y 500,000: 15
Producto de impares entre 1 y 500,000: 105
Producto de impares entre 1 y 500,000: 945
Producto de impares entre 1 y 500,000: 10395
Producto de impares entre 1 y 500,000: 135135
Producto de impares entre 1 y 500,000: 2027025
Producto de impares entre 1 y 500,000: 34459425
Producto de impares entre 1 y 500,000: 654729075
Producto de impares entre 1 y 500,000: 13749310575
Producto de impares entre 1 y 500,000: 316234143225
Producto de impares entre 1 y 500,000: 7905853580625
Producto de impares entre 1 y 500,000: 213458046676875
Producto de impares entre 1 y 500,000: 6190283353629375
Producto de impares entre 1 y 500,000: 191898783962510625
Producto de impares limitado debido a tamaño: 6332659870762850625
Calculos completadps en paralelo.
```

Ejercicio 4:

Encuentra el/los errores del siguiente código, explica porqué son errores y escribe el código correcto. El padre debería preguntar qué tal y el hijo contestar "bien".

- Tenemos que poner \\n en lugar de \n.
- La línea que contiene getpid() está cortada en mitad de la palabra.
- Faltan saltos de línea.
- Imprime mensajes del proceso padre mientras se está en el proceso hijo

cfgs@LAB-35-PC13: ~/Procesos

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

int main() {
        int pid;

        // Crear un proceso hijo
        pid = fork();

        if (pid == -1) {
                // Error al crear el proceso hijo
                perror("fork failed");
                exit(1);
        }

        // Mensaje común a ambos procesos
        printf("Hola, ¿qué tal? Todo bien, gracias.\n");

        if (pid == 0) {
                // Proceso hijo
                printf("\tSoy el proceso hijo, mi PID es %d\n", getpid());
                printf("Estoy bien, gracias por preguntar\n");
        } else {
                // Proceso padre
                printf("Soy el proceso padre, mi PID es %d\n", getpid());
        }

        return 0;
}
```

```
cfgs@LAB-35-PC13:~/Procesos$ vim codigoCorrecto.c
cfgs@LAB-35-PC13:~/Procesos$ gcc -o codigoCorrecto codigoCorrecto.c
cfgs@LAB-35-PC13:~/Procesos$ ./codigoCorrecto
Hola, ¿qué tal? Todo bien, gracias.
Soy el proceso padre, mi PID es 8422
Hola, ¿qué tal? Todo bien, gracias.
        Soy el proceso hijo, mi PID es 8423
Estoy bien, gracias por preguntar
```