

MEMORIA PRÁCTICA 1

Ejercicio4:

En el apartado A se realizan 3 forks y cada fork es realizado tanto por el padre como por el hijo, luego finalmente quedarían 8 procesos ejecutándose. Además, no hay ningún wait() así que los procesos hijos se quedan huérfanos, y más tarde se quedarían zombies, ocupando espacio en la tabla de procesos (los linux modernos eliminan automáticamente los procesos zombies).

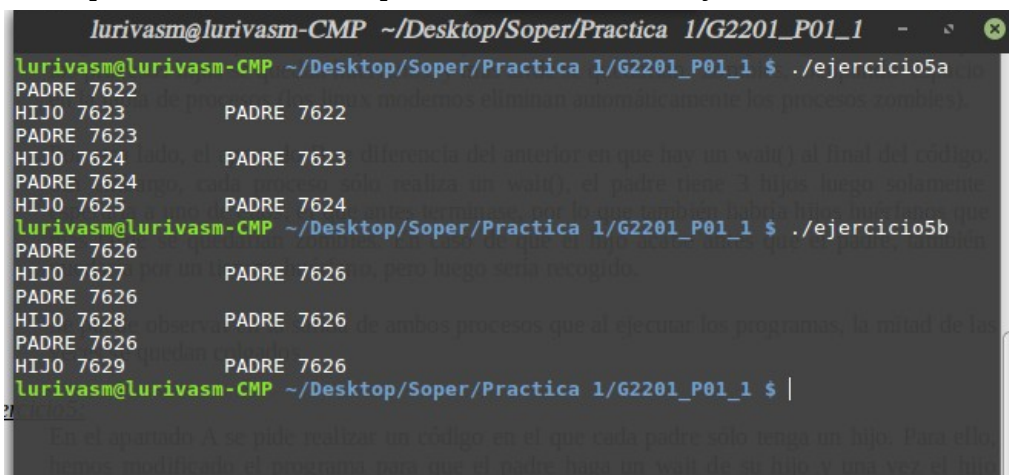
Por otro lado, el apartado B se diferencia del anterior en que hay un wait() al final del código. Sin embargo, cada proceso sólo realiza un wait(), el padre tiene 3 hijos luego solamente esperaría a uno de ellos, el que antes terminase, por lo que también habría hijos huérfanos que más tarde se quedarían zombies. En caso de que el hijo acabe antes que el padre, también quedaría por un tiempo huérfano, pero luego sería recogido.

Se puede observar en la salida de ambos procesos que al ejecutar los programas, la mitad de las veces se quedan colgados.

Ejercicio5:

En el apartado A se pide realizar un código en el que cada padre sólo tenga un hijo. Para ello, hemos modificado el programa para que el padre haga un wait de su hijo y una vez el hijo llegue, salga del bucle. Eso se ejecuta 3 veces, luego quedará el padre, hijo, nieto y bisnieto.

En el apartado B se pide que sea el padre el que tiene los hijos y que ningún hijo puede tener más hijos, para ello hacemos que el padre espere al hijo y que el hijo haga un break para finalizar, repitiéndose 3 veces. Se pueden observar ambos ejercicios en la foto:



```
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 - s x
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 $ ./ejercicio5a
PADRE 7622
HIJO 7623          PADRE 7622
PADRE 7623
HIJO 7624          PADRE 7623
PADRE 7624
HIJO 7625          PADRE 7624
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 $ ./ejercicio5b
PADRE 7626
HIJO 7627          PADRE 7626
PADRE 7626
HIJO 7628          PADRE 7626
PADRE 7626
HIJO 7629          PADRE 7626
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 $ |
```

Ejercicio 6:

En el ejercicio 6 hemos podido comprobar que efectivamente un padre no tiene acceso a los datos de memoria de su hijo pues al hacer el fork(), el hijo es una copia exacta del padre pero ahora sus datos tienen una memoria propia que no tiene nada que ver con la del padre. Para liberar la memoria, cada proceso debe llamar a un free luego puede hacerse dentro de cada

proceso o en el espacio en el que los dos procesos ejecutan lo mismo. En esta imagen se observa que el padre no accede, sólo podría mediante tuberías:

```
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 - [x]
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 $ ./ejercicio6
Padre esperando

Hijo empieza
Introduce un nombre: Lucia
Hijo imprime: Lucia

Hijo termina

Padre imprime:
Padre termina

lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 $ |
```

Ejercicio 8:

En este ejercicio, llamábamos mediante un `exec` a diferentes programas pasándolos como parámetros a la función principal, creando un `fork` por cada programa que pasamos. Podemos ver los siguientes ejemplos:

```
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 - [x]
ejercicio12b  ejercicio4b  ejercicio5a.c  ejercicio6.o  Makefile
ejercicio12b.c ejercicio4b.c  ejercicio5a.o  ejercicio8     Memoria.odt
ejercicio12b.o ejercicio4b.o  ejercicio5b    ejercicio8.c   Nombres.txt
ejercicio13   ejercicio4.c  ejercicio5b.c  ejercicio8.o
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 $ ./ejercicio8 ls -
vp
ejercicio12a  ejercicio13.c  Ejercicio4.c~  ejercicio5b.o  ejercicio9
ejercicio12a.c ejercicio13.o  ejercicio4.o  ejercicio6     ejercicio9.c
ejercicio12a.o ejercicio4     ejercicio5a    ejercicio6.c   ejercicio9.o
ejercicio12b  ejercicio4b    ejercicio5a.c  ejercicio6.o   Makefile
ejercicio12b.c ejercicio4b.c  ejercicio5a.o  ejercicio8     Memoria.odt
ejercicio12b.o ejercicio4b.o  ejercicio5b    ejercicio8.c   Nombres.txt
ejercicio13   ejercicio4.c  ejercicio5b.c  ejercicio8.o
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 $ ./ejercicio8 du l
s -lp
288
ejercicio12a  ejercicio13.c  Ejercicio4.c~  ejercicio5b.o  ejercicio9
ejercicio12a.c ejercicio13.o  ejercicio4.o  ejercicio6     ejercicio9.c
ejercicio12a.o ejercicio4     ejercicio5a    ejercicio6.c   ejercicio9.o
ejercicio12b  ejercicio4b    ejercicio5a.c  ejercicio6.o   Makefile
ejercicio12b.c ejercicio4b.c  ejercicio5a.o  ejercicio8     Memoria.odt
ejercicio12b.o ejercicio4b.o  ejercicio5b    ejercicio8.c   Nombres.txt
ejercicio13   ejercicio4.c  ejercicio5b.c  ejercicio8.o
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 $ |
```

Ejercicio 9:

Podemos ver 3 ejemplos de ejecución de tuberías entre procesos:

```
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 - [x]
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 $ ./ejercicio9
Introduce el primer operando: 2

Introduce el segundo operando: 3
Hijo 1 recibiendo datos
  Hijo 1 devuelve : 2.000 elevado a 3.000 = 8.000
Hijo 2 recibiendo datos
  Hijo 2 devuelve : factorial(2.000) entre 3.000 = 0.667
Hijo 3 recibiendo datos
  Hijo 3 devuelve : 2.000 sobre 3.000 = 0.000
Hijo 4 recibiendo datos
  Hijo 4 devuelve : abs(2.000 + 3.000) = 5.000
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 $ |
```

En este ejemplo, tras introducir los operandos, el padre los envía a los hijos y cada hijo devuelve el resultado de una operación.

En este ejemplo, tras

```
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 -  
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 $ ./ejercicio9  
Introduce el primer operando: 30  
  
Introduce el segundo operando: 50  
Hijo 1 recibiendo datos  
Hijo 1 devuelve : 30.000 elevado a 50.000 = 7178979876918526246008683463  
0974922672176658256911143378522493092565614592  
Hijo 2 recibiendo datos  
Hijo 2 devuelve : factorial(30.000) entre 50.000 = 530505719624382091399  
2071643136.000  
Hijo 3 recibiendo datos  
Hijo 3 devuelve : 30.000 sobre 50.000 = 0.000  
Hijo 4 recibiendo datos  
Hijo 4 devuelve : abs(30.000 + 50.000) = 80.000  
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 $ |
```

```
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 -  
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 $ ./ejercicio9  
Introduce el primer operando: 1000  
  
Introduce el segundo operando: 2  
Hijo 1 recibiendo datos  
Hijo 1 devuelve : 1000.000 elevado a 2.000 = 1000000.000  
Hijo 2 recibiendo datos  
Hijo 2 devuelve : factorial(1000.000) entre 2.000 = inf  
Hijo 3 recibiendo datos  
Hijo 3 devuelve : 1000.000 sobre 2.000 = 499500.000  
Hijo 4 recibiendo datos  
Hijo 4 devuelve : abs(1000.000 + 2.000) = 1002.000  
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 $ |
```

Ejercicio 12:

En este ejercicio se pide calcular los 10.000 primeros primos mediante 100 procesos y 100 hilos para ver con cual de los dos se tarda menos.

```
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 -  
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 $ ./ejercicio12a 10  
000  
Has tardado 1.177458 segundos  
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 $ ./ejercicio12b 10  
000  
Has tardado 0.028007 segundos  
lurivasm@lurivasm-CMP ~/Desktop/Soper/Practica 1/G2201_P01_1 $ |
```

En la imagen el primero corresponde a 100 forks mientras que el 12b corresponde a 100 hilos, se puede observar que los hilos tardan mucho menos en calcular los 10.000 primeros primos. Esto se debe a que un hilo no necesita hacer copia de ningún padre y a la hora de eliminarlo tampoco ha necesitado memoria por separado. Para medirlo con más exactitud, lo hemos ido incrementado a medida que se lanzaban los hilos/procesos pues al lanzarlos todos al mismo tiempo, como estamos en linux y los hilos son a nivel de usuario, deberían repartirse el tiempo entre ellos.