



Memoria Práctica 1 - Sistemas Operativos

Manuel Cintado y Manuel Suárez Román

Doble Grado Matemáticas e Ingeniería Informática

Grupo 2202

Contenido

Introducción	3
Desarrollo	3
Ejercicio 3	3
Ejercicio 4	5
Ejercicio 5	5
Ejercicio 9	5

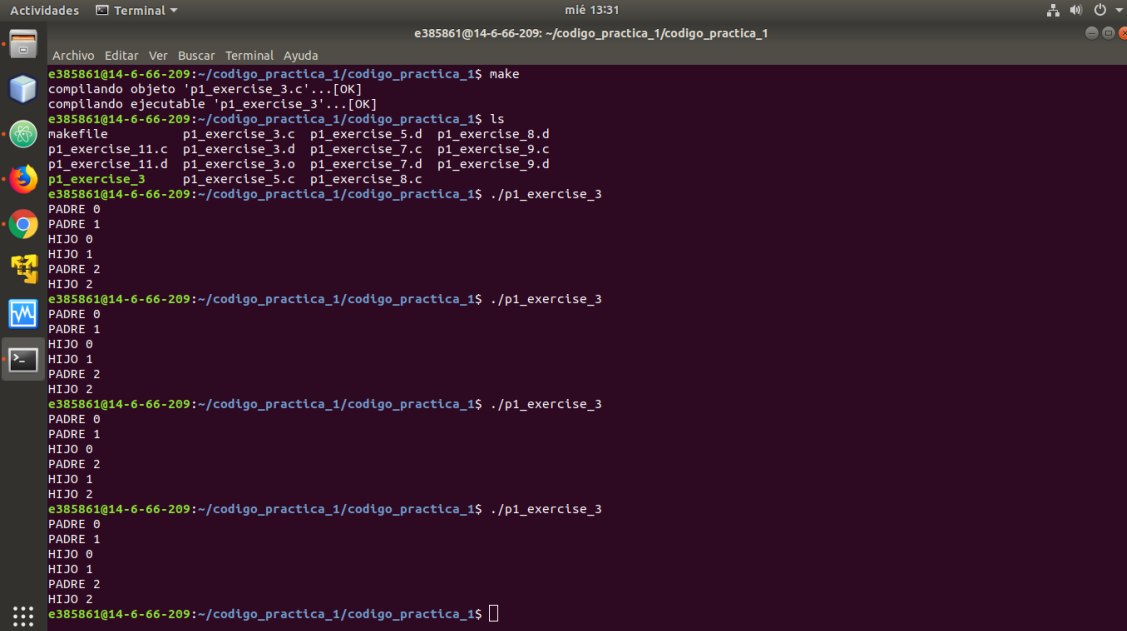
Introducción

En esta memoria, para no hacerla más extensa de la cuenta, hemos tomado la medida de solo incluir aquellos ejercicios que nos plantean cuestiones que son necesarias de resolver de una manera más teórica, excluyendo el adjuntar código pues consideramos que esto únicamente sobrecargaría innecesariamente el volumen de la memoria. En cualquier caso, se adjuntan en la carpeta “code” todos los archivos de código empleados en la realización de la práctica nombrados bajo el mismo patrón que se nos proporcionó con los ficheros base, lo que facilita enormemente la identificación del fichero de código correspondiente a cada ejercicio.

Desarrollo

Ejercicio 3

- a) No podemos determinar a priori el orden en el que se imprimirá por pantalla cada una de las instrucciones, pues depende de qué hilo se esté ejecutando antes y tanto los hijos padres como los hijos se ejecutan de manera concurrente por lo tanto no se puede saber cuál imprimirá por pantalla antes pues eso depende de la velocidad relativa de los procesos tal y como se puede comprobar al realizar varias ejecuciones del programa:



```
Actividades Terminal mié 13:31
e385861@14-6-66-209: ~/codigo_practica_1/codigo_practica_1

e385861@14-6-66-209:~/codigo_practica_1/codigo_practica_1$ make
compilando objeto 'p1_exercise_3.o'...[OK]
compilando ejecutable 'p1_exercise_3'...[OK]
e385861@14-6-66-209:~/codigo_practica_1/codigo_practica_1$ ls
makefile p1_exercise_3.c p1_exercise_5.d p1_exercise_8.d
p1_exercise_11.c p1_exercise_3.d p1_exercise_7.c p1_exercise_9.c
p1_exercise_11.d p1_exercise_3.o p1_exercise_7.d p1_exercise_9.d
p1_exercise_3 p1_exercise_5.c p1_exercise_8.c
e385861@14-6-66-209:~/codigo_practica_1/codigo_practica_1$ ./p1_exercise_3
PADRE 0
PADRE 1
HIJO 0
HIJO 1
PADRE 2
HIJO 2
e385861@14-6-66-209:~/codigo_practica_1/codigo_practica_1$ ./p1_exercise_3
PADRE 0
PADRE 1
HIJO 0
HIJO 1
PADRE 2
HIJO 2
e385861@14-6-66-209:~/codigo_practica_1/codigo_practica_1$ ./p1_exercise_3
PADRE 0
PADRE 1
HIJO 0
PADRE 2
HIJO 1
HIJO 2
e385861@14-6-66-209:~/codigo_practica_1/codigo_practica_1$ ./p1_exercise_3
PADRE 0
PADRE 1
HIJO 0
HIJO 1
PADRE 2
HIJO 2
e385861@14-6-66-209:~/codigo_practica_1/codigo_practica_1$
```

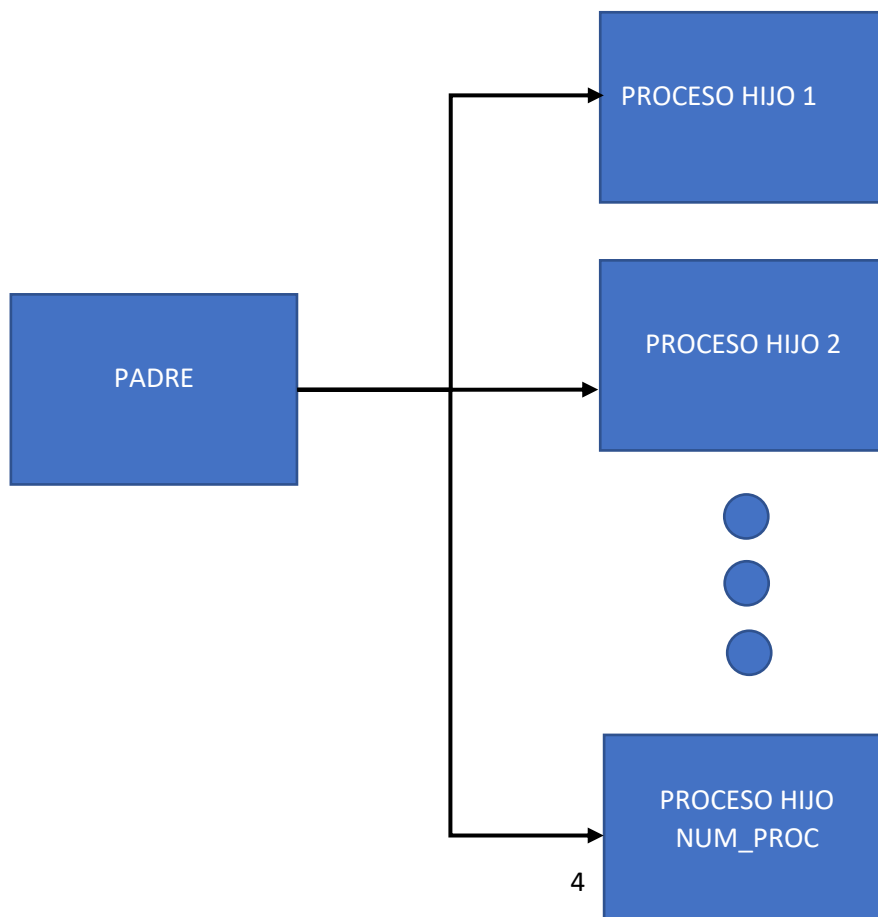
a) Simplemente hay que cambiar la función printf de la línea 23 por:

```
printf ("PPID: %d, PID: %d\n", (long)getppid (), (long)getpid ());
```

Lo que arroja los siguientes resultados de ejecución:

```
Actividades Terminal miércoles 13:50
e385861@14-6-66-209: ~/codigo_practica_1/codigo_practica_1
Archivo Editar Ver Buscar Terminal Ayuda
e385861@14-6-66-209:~/codigo_practica_1/codigo_practica_1$ ./p1_exercise_3
PADRE 0
PPID: 0, PID: 8230
PADRE 1
PPID: 8230, PID: 8231
PADRE 2
PPID: 8230, PID: 8232
e385861@14-6-66-209:~/codigo_practica_1/codigo_practica_1$
```

b) En este caso el programa crea un número de hijos igual al valor de NUM_PROC, lo que arroja el siguiente árbol de procesos:



Ejercicio 4

- a) Se puede comprobar que el proceso padre finaliza antes que los procesos hijos, pues el programa proporcionado no dispone de ninguna instrucción que permita esperar al padre a la finalización de los procesos hijos. De esta manera, el proceso padre se dedica a crear hijos mientras que, tras haberlos creado todos, el padre finaliza sin haber acabado los últimos hijos creados al no existir alguna instrucción que permita esto.
- b) En este caso únicamente hay que añadir un `sleep` que permite que el programa espera para que de esta manera no queden procesos hijos sin finalizar
- c) En este apartado, únicamente es necesario cambiar el código para que, en el proceso padre se espere a que el hijo finalice y así pueda acabar su ejecución. En caso de que sea el proceso hijo el que estamos tratando, proseguirá con la ejecución del bucle.

Ejercicio 5

- a) Al ejecutar el programa, no se imprime correctamente la frase que debería haberse copiado en el `strcpy`, pues esta función se ha ejecutado únicamente en el proceso hijo, y no en el padre. Al crear un hijo, se crea un proceso que contiene las mismas variables que se tenían en el momento de ejecución, pero a partir de ese momento, padre e hijo siguen ejecuciones distintas, es decir, que, si se modifican las variables en un hijo, salvo que lo forcemos, estas no se verán cambiadas en el padre. Por eso al imprimir por pantalla el valor de la frase dentro del padre aparece vacía, pues ha sido en el hijo donde se ha editado el valor de la frase.

Podemos comprobar esto si introducimos el `printf` dentro del condicional que se ejecuta cuando el `pid` es 0 (cuando estamos en el proceso hijo), pues ahí sí que imprime por pantalla correctamente la frase

- b) Es necesario añadir un `free(sentence)` en ambos condicionales, tanto en el hijo como en el padre, pues en caso contrario, se liberaría la memoria correspondiente al `char*` de uno de los procesos, pero no del otro, es decir, si tuviéramos el `free(sentence)` en el condicional de `pid == 0` por ejemplo, liberaremos el espacio de memoria que se ha reservado para `sentence` en el momento en el que se ha hecho el `fork` para el hijo, sin embargo, seguiría quedando sin liberar la memoria que tiene la variable `sentence` en el proceso padre. Por eso es necesario liberar memoria en ambos.

Ejercicio 9

En este ejercicio, que hemos decidido abordar desde una perspectiva de la programación modular, en primer lugar, pasamos a crear el `struct` requerido, que es bastante simple al solo poseer dos elementos tal y como se nos dice en el enunciado, a continuación, creamos la función que se encargará de obtener el resultado de la exponenciación. Esta función debe por

lo tanto recibir el elemento del array que contiene la información pertinente a ese hilo y lo almacena en el parámetro de solución

Tras esto pasamos a la función *main* en la que, a mediada que vamos creando los hilos, con *pthread_create*, llamamos a la función enviando el parámetro adecuado

Por último, solo tenemos que esperar a que todos los hilos terminen para que no se produzcan errores inesperados con el uso de *pthread_join* e imprimimos por pantalla los resultados