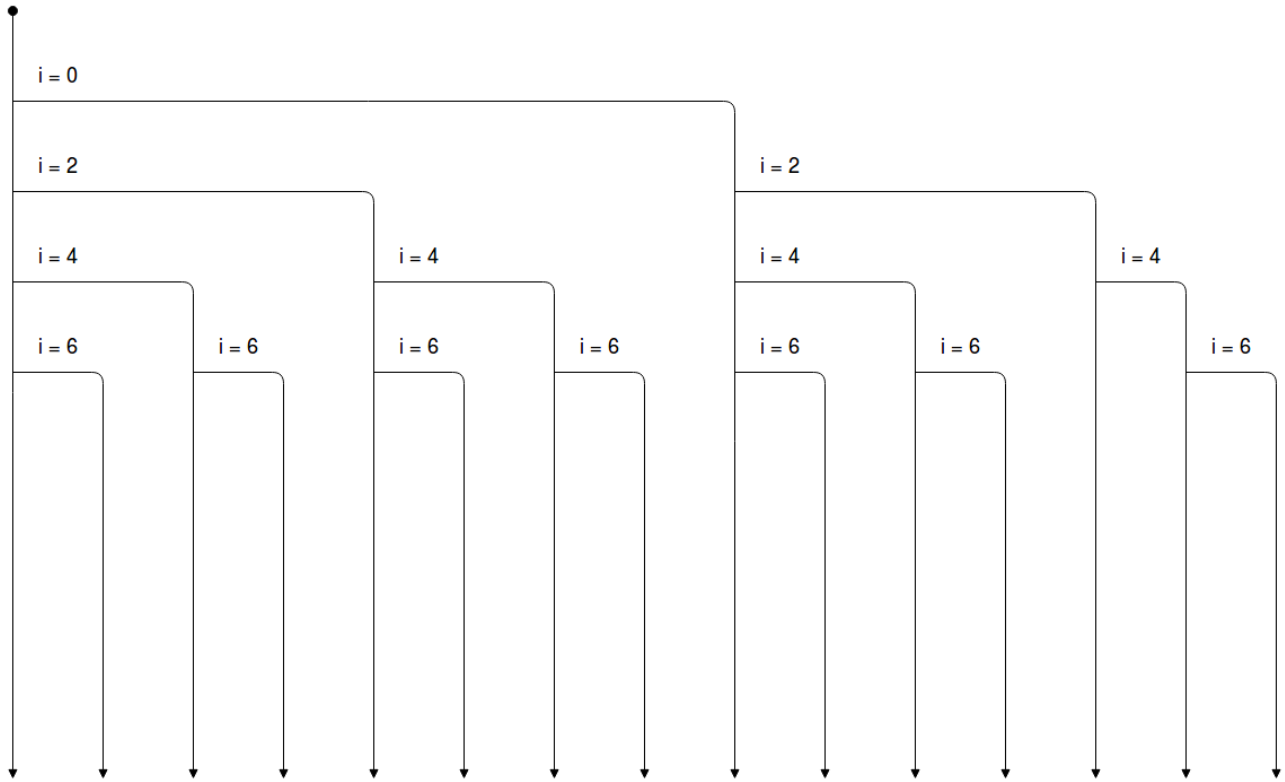
Javier Delgado del Cerro y Javier López Cano
Pareja 6

MEMORIA PRÁCTICA 1

Ejercicio 4:

a)



En este caso, con el código que se nos da, cada vez que en el bucle se llega a un número par, los procesos crean un hijo mediante un *fork()* y asi sucesivamente hasta que llega a *i = 6*, que el bucle acaba. Según el procedimiento del código, en este caso todos los procesos se quedan huerfanos (si acaban después de su padre) y finalmente zombies cuando el padre sale del bucle, puesto que en ningún momento del código se realiza ningún *wait* que espere a los hijos.

b)

En este caso, sin embargo, no todos los procesos se quedan zombies porque el codigo tiene un *wait* que hace que cada proceso espere al primero de sus hijos. De todos modos como muchos de los procesos tienen mas de un hijo, seguirá habiendo muchos procesos que quedaran zombies.
Por ejemplo, el padre ha creado cuatro hijos, con lo que tres de ellos acabarán zombies, pues el padre no recoge su exit. El primer hijo en ser creado crea otros tres procesos, por lo que de nuevo, dos de ellos acabarán zombies, los que más tarden en acabar, y lo mismo pasa con el segundo hijo en ser creado (el hijo del proceso principal para i=2), que crea otros dos procesos con lo que uno de ellos, el más lento, acabará zombie.
Todos estos procesos que quedan zombies pueden quedar también huérfanos si su padre acaba antes que ellos, y por tanto imprimirán 1 como pid del padre. Un vez acaban, quedan zomibies hasta que el sistema operativo los elimina.

Javier Delgado del Cerro y Javier López Cano
Pareja 6

Ejercicio 5:

a)
En primer lugar cambiamos la condicion del if poniendo != en lugar de ==, como se nos pedía en el enunciado. Mediante un if comprobamos si el proceso es padre o no, y, si lo es, realizamos un wait para que espere a que su hijo acabe y, una vez su hijo ha acabado, hacemos que el padre salga del bucle para evitar crear mas hijos.

b)
En primer lugar cambiamos la condicion del if poniendo != en lugar de ==, como se nos pedía en el enunciado. Tras esto modificamos el código haciendo que, antes de hacer el fork, se comprueba que el pid guardado no es 0, de forma que solo el padre pueda crear nuevos procesos. Así conseguimos que todos los hijos completen el bucle sin hacer ningún fork.

En el caso de que el proceso sea el padre, hacemos que espere a que su hijo acabe mediante un *waitpid*, de forma que el proceso sigue ejecutándose y generando el resto de hijos.

Ejercicio 6:

Una vez que el proceso hijo ha capturado el texto y ha acabado, intentamos imprimir en el proceso padre el texto que el proceso hijo ha guardado en el struct, sin embargo no obtenemos nada.
Usando valgrind, probamos a liberar la memoria en el if del proceso padre unicamente (pid > 0), en el if del proceso hijo unicamente (pid==0), o en el main, antes del *exit(EXIT_SUCCESS)*, y podemos ver que la unica forma de no perder memoria es liberándola fuera de los condicionales, antes del *exit*, o bien dentro de los dos condicionales, de forma que se libere la memoria tanto en el padre como en el hijo.

Esto se debe a que al llamar a *fork*, se crea una copia exacta e independiente del proceso que estamos ejecutando, con las variables incluidas. Por tanto, el hijo y el padre estan accediendo a posiciones de memoria distintas. Esto es lo que nos permite, como se ve en el fichero *Ejercicio6_dem*, acceder a una copia de la informacion que el padre ha guardado en el struct antes de crear el hijo.

Ejercicio 8:

Para facilitar la tarea pedida en este ejercicio decidimos crear una función distinta para cada uno de los *exec* (*execl*, *execv*…). Además, en los casos en los que hay que introucir el path puesto que la función no lo da automáticamente (*execl* y *execv*) tuvimos que usar dos paths distintos */bin/* y */usr/bin/* para evitar errores por la localización de los comandos que se nos pedía ejecutar.

Javier Delgado del Cerro y Javier López Cano
Pareja 6

Ejercicio 9:

En primer lugar creamos las funciones para llevar a cabo las operaciones requeridas.
Tras esto, en el main inicializamos ocho pipes, dos para cada uno de los hijos de modo que la comunicación de cada uno de ellos con el padre sea bidireccional. Entonces pedimos los dos operandos por pantalla y hacemos que el proceso padre escriba en cuatro de los pipes (uno por cada proceso hijo) los dos operandos. Luego vamos comprobando el pid de los distintos procesos de modo que cada uno de ellos realice una de las operaciones y escriba el resultado en el pipe correspondiente. Por último hacemos que el padre lea los resultados de las cuatro pipes en las que escribieron los hijos y los imprima por pantalla.

Ejercicio12:

Se obtiene un mejor rendimiento con el programa de los hilos puesto que estos no tienen que incializar bloque de control de proceso ni copiar todas las variables del proceso padre, por lo que tardan considerablemente menos en su ejecución. Sin embargo, en este programa en concreto, aunque se ve que son más rápidos, la diferencia no es excesivamente apreciable.
Por poner algunos ejemplos, para 1000 procesos e hilos, la diferencia es de aproximadamente un milisegundo, y para N=10.000 la diferencia es de unos dos segundos.

Ejercicio13:

La forma de compartir información entre hilos es empleando variables globales tal como hemos realizado en el *Ejercicio13b.c* entregado.