Javier Delgado del Cerro y Javier López Cano
Pareja 6

MEMORIA PRÁCTICA 2

Ejercicio 2:

Generamos el programa pedido, en el que el padre crea un hijo que imprime “Soy el proceso hijo <PID>” y duerme 30 segundos, tras lo cual imprime “Soy el proceso hijo <PID> y ya me toca terminar”. Mientras, el padre duerme cinco segundos tras hacer el fork y después le manda una señal *SIGTERM*. Así, repite el proceso otras tres veces hasta haber creado cuatro hijos.

De esta forma, dado que el hijo no tiene declarada ninguna función de manejo para la señal *SIGTERM*, y puesto que el hijo duerme treinta segundos y el padre tan solo cinco, ninguno de los hijos va a ser capaz de imprimir el segundo mensaje, pues recibirán la señal mientras están dormidos y morirán en ese momento al usar el manejador de la señal por defecto *SIG_DFL*.

Ejercicio 4:

Para la realizacion de este ejercicio, simplemente hemos usado un blucle for en el que creamos los *N* hijos pedidos. De esta forma, cada vez que el padre crea un hijo, se bloquea llamando a *pause()*, y el hijo, en el caso de no ser el primero, manda una señal al hijo anterior para que éste termine, e imprime diez veces por terminal el mensaje “Soy <PID> y estoy trabajando”, esperando un segundo entre impresión e impresión. Una vez ha acabado, manda una señal al padre para que cree el nuevo hijo, y este nuevo hijo le destruya a él, mientras sigue imprimiendo el mismo mensaje.
Así, cuando ya ha creado los N procesos hijos, es el padre el que manda la señal al último hijo para que éste acabe, y entonces acaba el padre, tras haberse asegurado de que todos sus hijos han muerto.

Para que todos los hijos puedan saber el PID del anterior, usamos un array de enteros, que se duplica al hacer el fork.

Para despertar al padre, usamos un manejador que simplemente hace *return*, pues si usásemos una señal ignorada, el padre no se despertaria, y si no usásemos manejador, el padre moriría al recibir la señal primer hijo y no podría crear al resto.

Ejercicio 6

Apartado A:
Despues de hacer *fork*, en el proceso hijo, creamos dos máscaras de señales vacias: en la primera incluimos *SIGUSR1* y en la segunda *SIGUSR2* y *SIGALRM*. Entonces, establecemos una alarma dentro de cuarenta segundos, y en cada ciclo del *while* bloqueamos las tres señales al iniciarlo, usando *sigprocmask(SIG_BLOCK, &set,&oset)*, imprimimos los números, y antes de repetir el while, desbloqueamos las señales del segundo conjunto usando *sigprocmask(SIG_UNBLOCK, &set, &oset);* y dormimos tres segundos más.

De este forma, cuando el hijo recibe la señal, si esta imprimiendo, dicha señal se queda bloqueada hasta que acaba el bucle for y desactiva las dos señales de la segunda máscara. En ese momento, como el hijo no tiene un manejador de la señal *SIGALRM*, realiza la accion por defecto, que es terminar el proceso, *SIG_DFL*.

El padre solo tiene que hacer el *fork* y un *wait* para esperar a que el hijo termine antes de acabar él.

Apartado B:
En este caso, simplemente definimos tras el *fork* un manejador para la señal *SIGTERM* que imprime "Soy <PID> y he recibido la señal SIGTERM", y termina el proceso.

Así, el padre hace el *fork*, duerme cuarenta segundos, envia la señal y hace un *wait* para esperar al hijo antes de acabar.

Ejercicio 8

En este ejercicio solo hemos tenido que codificar en *Ejercicio8.c* y *Ejercicio8.h* la librería de semáforos pedida basándonos en el ejemplo del ejercicio siete anterior.

Comentar que en la función de *Crear_Semaforo* no inicializamos los semáforos, pues implicaba reservar memoria para un array de *unsigned short* que no podíamos liberar posteriormente de forma segura.

Hemos implementado también en el archivo *Ejercicio8.c* una función *test* que nos ha permitido probar dicha librería, ejecutando el programa del ejercicio siete usando nuestras funciones.

Ejercicio 9

En este ejercicio nueve, hemos intentado distribuir el trabajo en funciones de forma que el programa fuese lo más limpio y ordenado posible, para evitar posibles errores.

En primer lugar, el proceso padre se encarga de inicializar un semáforo para cada uno de los procesos, que usamos para gestionar el fichero de la cuenta de cada una de las cajas, y otro más, de sincronización, que utilizamos para saber qué proceso ha mandado la señal.

Una vez ha inicializado los semáforos, el padre inicializa dos manejadores de señales, *retirarParte* y *retirarTotal*, que atienden a *SIGUSR1* y *SIGUSR2* respectivamente, y crea un archivo *clientesCaja.txt* con 50 clientes de entre 0 y 300 euros para cada una de las cajas (usando la función *rellenar_fichero*), ademas de hacer el *fork* ara crear a los hijos.

Es entonces cuando los hijos empiezan a leer su fichero de caja, usando la función *mod_caja*, que protege el fichero de cada caja mediante los semáforos creados anteriormente, de forma que es imposible que dos procesos (el padre y el hijo) lo intenten leer o editar simultaneamente.

Una vez el hijo ha conseguido más de mil euros en su caja, o ha acabado, baja el semáforo de sincronización que hemos mencionado anteriormente, escribe su número de caja en el fichero *procesoEspera.txt* y manda la señal *SIGUSR1* o *SIGUSR2* al padre. Así, los manejadores de señal del padre llaman a *retirarParte* o a *retirarTotal*, que utilizan el archivo *procesoEspera.txt* y la función *mod_caja* para extraer todo el dinero, o sólo 900 euros de la caja que ha mandado la señal, haciendo posteriormente un up del semáforo.

De esta forma, es posible que un mismo proceso, al tener mas de mil euros, mande una vez la señal al padre, procese otro pago antes de que el padre atienda la señal, y por tanto, se quede bloqueado en el semáforo y vuelva a enviar posteriormente la señal. Sin embargo, dado que se pedía que aún despues de mandar la señal la caja siguiera funcionando, en nuestra función *mod_caja* tenemos en cuenta el valor de la caja antes de extraer el dinero, para prevenir posibles errores. Si no se hubiera pedido esto, lo más sencillo habría sido inicializar este semáforo a cero, de forma que el proceso se bloquease hasta que el padre hubiese atendido la señal.

Javier Delgado del Cerro y Javier López Cano
Pareja 6

Finalmente, hemos conseguido que el programa funcione correctamente, de forma que cada caja avisa por terminal cuando ha hecho una operación del dinero actual que tiene, y cuando ha acabado, y el padre imprime cuando ha sacado dinero de cada una de las cajas y, finalmente, indica el dinero total que ha obtenido.