PRÁCTICA 2 – SOPER
Pareja 1
Alba Ramos
Andrea Salcedo

Todos los ejercicios que hay que entregar se compilan haciendo make nombre_archivo.c (este nombre se indica al final de cada apartado). También podemos
compilar todos a la vez haciendo make.
Para ejecutarlos, se escribe ./nombre_archivo, sin el '.c'.

EJERCICIO 2

Cada hijo imprime "Soy el proceso hijo <PID>", y nunca imprimen el mensaje de que les toca terminar, porque como tienen que dormir más tiempo que el padre, el procesador le va a dar el turno de ejecución al padre cuando ellos se vayan a dormir. El padre dormirá el tiempo que necesite y matará a los hijos directamente, impidiendo que impriman el otro mensaje, puesto que se está ejecutando la rutina de tratamiento de señales por defecto. El tiempo que duerme el padre es menor que el que duermen los hijos, así que no puede ocurrir que el procesador le de ejecución al hijo mientras el padre duerme. (***ejercicio2.c***)

EJERCICIO 3

a) No. La llamada implica que se establece que el manejador para esa señal será el que indicamos en los argumentos, pero hasta que no llegue la señal no se va a ejecutar la función manejador. La señal puede tardar en llegar.

b) El printf del manejador aparece cuando hemos realizado "ctrl+c", que genera la señal SIGINT y entonces se ejecuta el manejador.

c) Se llama a la rutina de tratamiento por defecto, en la que normalmente se termina la ejecución del proceso, pero también se puede generar un fichero core en el que se almacena el contexto del proceso.

d) El programa nunca logra ejecutar el manejador para SIGKILL, y por tanto nunca imprime lo que tengamos dentro de él, además de mostrar un error de argumentos en la función sigaction. Esto ocurre debido a que SIGKILL es una señal que no puede ser capturada, ya que la señal nunca llega realmente al proceso, sino que se queda en el sistema operativo, puesto que implica la terminación inmediata del proceso.(***ejercicio3d.c***)

EJERCICIO 4

Para este programa, hemos utilizado una macro NUM_PROC para establecer cuántos procesos hijos participarán en la carrera, y poder editar esta cantidad fácilmente. También hemos creado dos manejadores que muestran un mensaje por pantalla, para ver que las

señales se enviaban y recibían bien en todo momento.
Primero hacemos una llamada a fork para que el padre genere un hijo, que será el gestor de la carrera. El gestor crea después a todos los participantes, esperando antes de crear al siguiente a que se le envíe la señal SIGUSR2 mediante pause. Los hijos se la envían al padre usando kill y getppid, y el gestor la recibe utilizando un manejador. Cuando ha recibido la señal de todos los hijos (es decir, fuera del bucle), el gestor envía una señal al padre. El padre está en pause hasta que recibe la señal del gestor. Cuando la recibe, manda SIGUSR1 a todo el grupo utilizando el parámetro 0 en kill. Como él también la recibe, indicamos en su manejador que debe ignorarla, mediante la macro SIG_IGN. Los participantes, cuando la reciben, la capturan y muestran un mensaje, y terminan. El gestor, cuando la recibe, espera a los participantes con wait y termina. Finalmente, el padre espera con wait al fin del gestor y termina. (***ejercicio4.c***).

EJERCICIO 6

a) La señal SIGALRM termina la ejecución de un proceso. Entonces, el hijo termina su ejecución cuando la señal le llega, ya que está desbloqueada. Esto es posible, ya que al salir del for, estamos desbloqueando las señales que haya dentro del conjunto, salvo SIGUSR2 que ha sido eliminada del mismo. El bucle que imprime se va a hacer siempre completo, aunque llegase la alarma en medio de su ejecución, puesto que esta señal no se desbloquea hasta el final del bucle. (***ejercicio6a.c***)

b) En este caso, el hijo tiene un manejador de SIGTERM, y cuando esta señal le llegue, se va a ejecutar el mismo, interrumpiendo el bucle que cuenta en el instante en el que la señal se reciba. El padre duerme durante 40 segundos con un sleep y después envía la señal al hijo, al que espera antes de terminar. (***ejercicio6b.c***)

EJERCICIO 8

Para resolver este problema, hemos hecho que el padre espere a tener los hijos creados antes de meterse a ejecutar el bucle infinito. Antes de ejecutar ese bucle también hemos añadido 2 manejadores: uno para SIGINT que enviará SIGTERM a todo el grupo cuando la señal sea recibida, y otro para esa señal SIGTERM en el padre, que hará que se liberen los recursos de los semáforos, se espere a todos los hijos y se termine la ejecución.
Hemos creado 3 semáforos:

- sem_recurso: controla el acceso al recurso tanto para lectura como para escritura
- sem_contador: controla el acceso al contador
- sem_lectores: actúa como contador

Los semáforos los hemos puesto como variables globales para facilitar el cierre de los mismos desde los manejadores. Los hemos inicializado a 1 (salvo el contador que está a 0) para que el primero que llegue pueda entrar. A la hora de comprobar el valor del contador, hay que hacer sem_getvalue para obtener el valor actualizado. (***ejercicio8.c***)

a) SI SECS=0 y N_READ=1, hay lecturas (del único hijo) y escrituras. Se produce primero escritura del padre porque el procesador de estos ordenadores da prioridad al padre, y luego se van intercalando lecturas (del único hijo) y escrituras del padre. Esto tiene

sentido, ya que cada vez que alguien termina de leer o escribir, libera el semáforo y se pone a la cola, y el siguiente entra al recurso.

b) Si SECS=1 y N_READ=10, el padre no escribirá hasta que no tenga creados los 10 hijos, por lo que primero habrá lecturas. Primero se crea el primer hijo, el cual solicita actualizar el contador y puede hacerlo porque no hay nadie más esperando. El contador pasa a valer 1, hace el up y lo deja libre. Ahora solicita la lectura del recurso, y se le concede porque no hay nadie. Se va a quedar esperando en el sleep de leer(), y el procesador va a darle el turno a otro hijo (porque una vez que lee uno, pueden leer todos). Cuando ya estan los 10 hijos, el padre va a solicitar el recurso. Entonces, al terminar de leer, los hijos van decrementando el contador, hasta que el último hijo deja libre el recurso. Ahora es el turno del padre, mientras los hijos están de nuevo en la cola, esperando para leer hasta que acabe el padre, y así sucesivamente.

c) Para SECS=0 y N_READ=10, el padre esperará a que todos los hijos estén creados, por tanto ocurren primero lecturas, y estas hacen que una vez que el primero lea, ya todos puedan hacerlo. Como al final del bucle no hay sleep, el lector que acaba de terminar vuelve a empezar instantáneamente el bucle while, actualizando el contador. Solo se le va a dar tiempo de ejecución a otro proceso durante el sleep de la función leer(). En este momento, los lectores que hayan acabado el while tampoco estarán durmiendo y volverán instantáneamente a comenzar, y en el sleep de leer() es cuando se les dará tiempo de ejecución. Es por eso por lo que realmente nunca va a llegar a 0 el contador, sin permitir que haya escrituras.

d) Ocurre lo mismo que en el apartado anterior, ya que dormir durante 0 segundos es lo mismo que quitar la llamada a sleep.

EJERCICIO 9

Para resolver este problema, hemos utilizado un array, cuenta de N_PROC posiciones (0, 1, … N_PROC-1) que se corresponden con el id de cada participante en la carrera. Lo hemos hecho utilizando calloc, porque queremos que las posiciones estén inicializadas con el valor 0. Cuando los hijos escriben su id en el fichero (debidamente protegido con un semáforo), el padre tendrá que leer cuántas veces se ha escrito este id, y actualizar el valor de cuenta[id]. De esta forma, es sencillo actualizarlo y controlar el id del proceso que haya ganado. Cuando hay un ganador, se deben terminar con SIGTERM los hijos, que cierran el semáforo. El padre también debe cerrarlo y también liberar la memoria dinámica reservada. Hemos podido comprobar que dependiendo del tiempo de ejecución, cualquiera de los hijos creados puede ganar la carrera, aunque mayormente ganaba más veces el primer proceso. Cada proceso empieza escribiendo su id, y cuando le toca dormir, se pone a la cola. Así, se realiza una ejecución circular. La victoria dependerá del tiempo de ejecución y ver que proceso hijo a escrito en más ocasiones su id.(***ejercicio9.c***)