

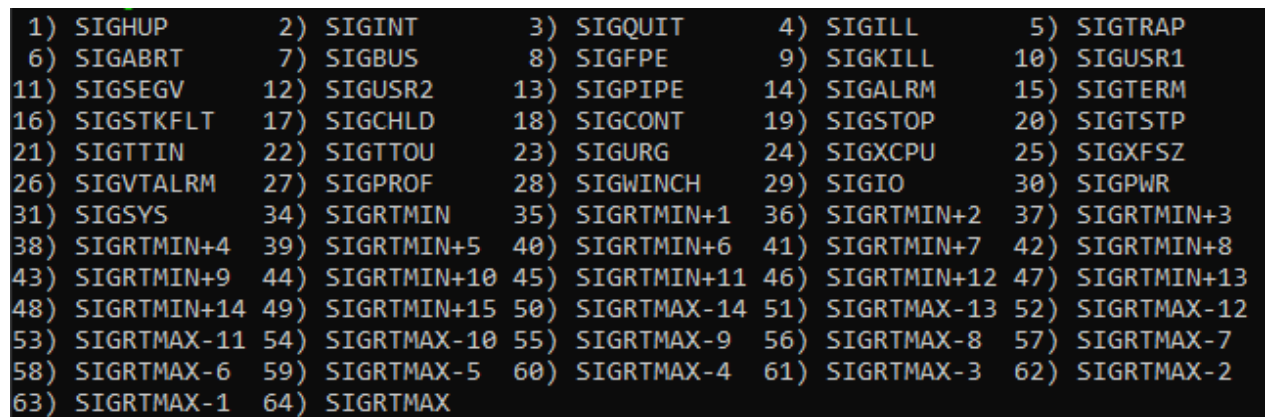
Sistemas Operativos, Práctica 2: Memoria

Leandro García y Fabián Gutiérrez (Pareja 02)

26 de marzo de 2020

Ejercicio 1

- a) Utilizando el comando `kill -l` se obtiene una lista de todas las señales que puede enviar la función `kill`.



1) SIGHUP	2) SIGINT	3) SIGQUIT	4) SIGILL	5) SIGTRAP
6) SIGABRT	7) SIGBUS	8) SIGFPE	9) SIGKILL	10) SIGUSR1
11) SIGSEGV	12) SIGUSR2	13) SIGPIPE	14) SIGALRM	15) SIGTERM
16) SIGSTKFLT	17) SIGCHLD	18) SIGCONT	19) SIGSTOP	20) SIGTSTP
21) SIGTTIN	22) SIGTTOU	23) SIGURG	24) SIGXCPU	25) SIGXFSZ
26) SIGVTALRM	27) SIGPROF	28) SIGWINCH	29) SIGIO	30) SIGPWR
31) SIGSYS	34) SIGRTMIN	35) SIGRTMIN+1	36) SIGRTMIN+2	37) SIGRTMIN+3
38) SIGRTMIN+4	39) SIGRTMIN+5	40) SIGRTMIN+6	41) SIGRTMIN+7	42) SIGRTMIN+8
43) SIGRTMIN+9	44) SIGRTMIN+10	45) SIGRTMIN+11	46) SIGRTMIN+12	47) SIGRTMIN+13
48) SIGRTMIN+14	49) SIGRTMIN+15	50) SIGRTMAX-14	51) SIGRTMAX-13	52) SIGRTMAX-12
53) SIGRTMAX-11	54) SIGRTMAX-10	55) SIGRTMAX-9	56) SIGRTMAX-8	57) SIGRTMAX-7
58) SIGRTMAX-6	59) SIGRTMAX-5	60) SIGRTMAX-4	61) SIGRTMAX-3	62) SIGRTMAX-2
63) SIGRTMAX-1	64) SIGRTMAX			

Figura 1: Lista de señales disponibles

- b) La señal `SIGKILL` se corresponde con el número 9, mientras que a `SIGSTOP` se le asigna el número 19.

Ejercicio 2

- b) Tras pasar la señal `SIGSTOP` de una terminal a otra, no es posible escribir en la terminal que recibe la señal. Una vez enviada la señal `SIGCONT`, dicha terminal reestablece su funcionamiento normal.

Ejercicio 3

- a) No, `sigaction` determina el comportamiento del proceso ante la recepción de una cierta señal; en este caso, indica la rutina de tratamiento de la señal (*manejador*).
- b) Puesto que no se utiliza la bandera `SA_NODEFER`, la rutina de tratamiento bloquea la señal que provocó su llamada (su *trigger*). En este caso, el *trigger* es `SIGINT`, por lo que *manejador* la bloquea.

- c) El `printf` de `manejador` se ejecuta, como es de esperarse, cuando el proceso recibe la señal `SIGINT` (Ctrl+C en la shell), y se muestra por pantalla tras ejecutar `fflush(stdout)`. Asimismo, el `printf` de `main` se muestra por pantalla automáticamente (esto es, sin necesidad de recibir `SIGINT`) al ejecutar el programa y luego de cada salida de `manejador`, puesto que la llamada a esta rutina desbloquea al proceso (que estaba dormido) y retoma el bucle `while`.

Ejercicio 4

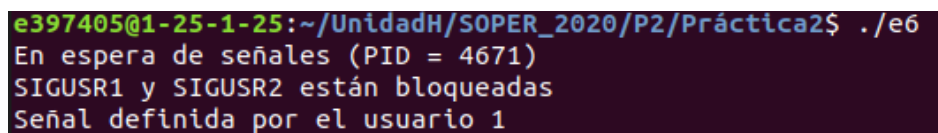
- a) Se lleva a cabo la acción por defecto asociada a dicha señal, que por lo general lleva a la terminación del proceso, y en algunos casos, la generación de un *core*.
- b) Haciendo sucesivas llamadas a `sigaction` para las señales entre 1 y 31, se observa que se produce un error para las señales 9 (`SIGKILL`) y 19 (`SIGSTOP`). De hecho, en el manual de `sigaction` se indica que `signum`, la señal a capturar, no puede ser ninguna de las anteriores. Por tanto, se ejecuta `manejador` para todas las señales salvo `SIGKILL` y `SIGSTOP` (lo que permite terminar este proceso con `kill` desde otra shell)

Ejercicio 5

- a) La gestión de la señal se hace *de facto* en `main`, específicamente entre las líneas 28 y 31.
- b) El uso de una variable global permite la comunicación entre `manejador` y `main`. En particular, permite que la primera “avise” a la segunda de que se recibió `SIGINT`. El uso de esta variable no supone un riesgo notable dado que no pueden surgir problemas de sincronización (en este caso).

Ejercicio 6

- a) Al enviar las señales `SIGUSR1` y `SIGUSR2` el programa no hace nada, puesto que estas están bloqueadas.
- b) Al finalizar la espera el programa termina con el mensaje mostrado en la imagen. El mensaje de final de programa no se imprime por pantalla ya que, al desactivarse la máscara, la señal `SIGUSR1` llega e interrumpe el proceso (y acaba terminando).



```
e397405@1-25-1-25:~/UnidadH/SOPER_2020/P2/Práctica2$ ./e6
En espera de señales (PID = 4671)
SIGUSR1 y SIGUSR2 están bloqueadas
Señal definida por el usuario 1
```

Figura 2: Resultado de la ejecución

Ejercicio 7

- a) Al llegar la señal `SIGALRM` durante la cuenta, se ejecuta `manejador` (imprime el mensaje y finaliza), lo que en ausencia de señales (originadas por otro proceso) ocurriría al cabo de `SECS` segundos con `alarm`.
- b) Como no hay una rutina de tratamiento de la señal `SIGALRM`, al recibir esta (tras los `SECS` segundos de `alarm`) se ejecuta la rutina por defecto, que termina el proceso.

Ejercicio 8

- a) Para tener un buen control sobre la recepción y tratamiento de señales se utilizaron múltiples máscaras, todas inicializadas antes del lanzamiento de los hijos (además de las estructuras `sigaction` para los manejadores). Asimismo, a la hora de indicar el manejador para una cierta señal, se bloquea previamente dicha señal con `sigprocmask` (esta será desbloqueada posteriormente o bien por otro `sigprocmask` o bien por `sigsuspend` en caso de que haya que esperarla).

Antes de lanzar los N hijos, se define la rutina de tratamiento de `SIGUSR2` para que el proceso padre pueda recibir las señales de sus hijos de forma segura. En cuanto a los hijos, tras realizar el trabajo indicado, se define la rutina de tratamiento de `SIGTERM` y se llama a `sigsuspend` con la máscara apropiada para esperar la señal del padre. Posteriormente no se realiza un `sigprocmask` que desbloquee definitivamente la señal (tras haber sido bloqueada antes del `sigaction`) puesto que el manejador de esta finaliza el proceso.

Por otra parte, tras lanzar los procesos hijos, el proceso padre define su rutina de tratamiento de `SIGALRM` para luego programar la alarma y esperarla mediante `sigsuspend`. Finalmente, una vez despierto por la alarma, el proceso padre envía la señal `SIGTERM` a todos sus hijos y posteriormente los espera.

- b) Para valores de N grandes (a partir de aproximadamente 50), se obtiene una cuenta cercana pero inferior a N, cuenta que puede variar con cada ejecución. Esto ocurre ya que las señales enviadas por los hijos pueden perderse debido a que, mientras el padre está esperando a `SIGALRM`, el resto de señales están bloqueadas, y solo se almacenan los tipos de señales pendientes, no se lleva la cuenta de cuántas hay de cada tipo (como en una máscara, cada señal se asocia a un bit). Entonces, al despertar, el padre trata una única señal `SIGUSR2`, aunque pudiera haber recibido y bloqueado varias durante la espera.

Cabe destacar que, según este análisis, el número de señales perdidas debería depender del valor de T el tiempo de espera. Sin embargo, el proceso padre tarda más en llegar a la espera en función del número de hijos que ha de crear, por lo que el efecto de T es despreciable (al menos para lo que tardan los hijos en hacer el trabajo).

Una alternativa al diseño implementado sería hacer que el proceso padre despierte de su espera a la alarma tanto con `SIGALRM` como con `SIGUSR2`. El manejador de la primera cambiará una variable global `recibida_alarma`. Por tanto, si el proceso padre despierta con `SIGUSR2`, incrementará el contador y al comprobar que `recibida_alarma` vale 0 volverá a dormirse a la espera de la alarma. Como en la implementación anterior, `SIGALRM` estará bloqueada en caso de recibirse `SIGUSR2` durante la ejecución de su manejador, y al volver a llamarse a `sigsuspend` detectará que la alarma estaba pendiente y volverá a despertar (haciendo `recibida_alarma = 1`).

```
/* Bloquear SIGALRM y SIGUSR2 y definir manejadores */
do {
    sigsuspend(&wait_alarm_usr2);
} while (!recibida_alarma);
/* Desbloquear SIGALRM y SIGUSR2 */
```

Con esta alternativa, la pérdida de las señales de usuario no ocurre por el solapamiento de señales pendientes durante la espera de la alarma (por lo que el valor de T no influye). Sin embargo, este fenómeno sí puede ocurrir en caso de que llegen múltiples señales `SIGUSR2` durante la ejecución del manejador correspondiente, ya que este, por defecto, bloquea el tipo de señal que atiende.

Ejercicio 9

Sí, de hecho, puede hacerse como pronto justo después de crear el semáforo, ya que no vuelve a accederse a él a través de su nombre.

Ejercicio 10

- a) Cuando se envía la señal SIGINT se muestra por pantalla el mensaje **Fin de la espera**, que se encuentra después de la llamada a `sem_wait`. Esto ocurre ya que al hacer `sem_wait` sobre un semáforo con valor 0, el proceso se bloquea hasta que el valor del semáforo es mayor que cero o hasta que un manejador de señal interrumpe la llamada.
- b) Si se decide ignorar la señal enviada al proceso, este no entra en el manejador y permanece bloqueado en el semáforo.
- c) Se debe aplicar una máscara de señales al fragmento de código del semáforo para evitar que estas interrumpan el funcionamiento de `sem_wait`.

Ejercicio 11

```
/* Código A y F: nada*/
```

```
/* Código B */  
sem_post(sem1);  
sem_wait(sem2);
```

```
/* Código C */  
sem_post(sem1);
```

```
/* Código D */  
sem_wait(sem1);
```

```
/* Código E */  
sem_post(sem2);  
sem_wait(sem1);
```

El proceso padre ha de esperar que el hijo imprima 1 (`sem_wait(sem1)` en D). Por su parte, el hijo debe avisar que imprimió 1 (`sem_post(sem1)` en B) y esperar a que el padre imprima 2 (`sem_wait(sem2)` en B). Luego el padre, una vez desbloqueado, imprime 2, avisa que lo hizo y queda a la espera de que el hijo imprima 3 (`sem_post(sem2)` y `sem_wait(sem1)` en E). Finalmente, el hijo imprime 3, lo avisa (`sem_post(sem1)` en C) y continúa su trabajo sin problemas de sincronización, al igual que el padre tras ser desbloqueado.

Ejercicio 12

En esta modificación del ejercicio 8 se añade a los hijos un sistema de lectura y escritura en un fichero de datos regulado por un semáforo. El proceso padre utiliza también este semáforo para acceder al fichero a realizar las comprobaciones necesarias.

A estos cambios hay que añadir la sustitución de la llamada a `sigsuspend` por un bucle `while` que, además de esperar a la señal `SIGALRM`, atiende la llegada de señales `SIGUSR2` (como en la segunda solución propuesta en el ejercicio 8. Esto permite que el padre, en lugar de contar las señales `SIGUSR2` que recibe, lea la cuenta del fichero cada vez que recibe esta señal y soluciona el problema de la pérdida de señales bloqueadas, ya que con atender una de ellas el padre lee la cuenta actualizada.

Ejercicio 13

Para la resolución de este ejercicio se ha implementado un semáforo adicional que se inicializa a 0. Cuando el último hijo escribe en el documento, este realiza un `sem_post` a este nuevo semáforo, dejando continuar al proceso padre.

Esta nueva funcionalidad sustituye a todas las señales `SIGALRM` (al no poder haber pérdidas de señales se garantiza la finalización del proceso) y `SIGUSR2`, simplificando así el código y evitando que el padre necesite verificar el estado del fichero constantemente.

Otra alternativa es el uso de semáforos N-arios. Cada hijo hará un `sem_post` al terminar y el padre hará N `sem_wait` en bucle. Esta solución es similar al funcionamiento de las versiones anteriores del problema: cada hijo avisa al padre y el padre espera a recibir N avisos antes de seguir.

Ejercicio 14

- b) Si se establecen las macros `N_READ 1` y `SECS 0`, el programa alternara una lectura y una escritura en bucle hasta recibir la señal de parada. Esto se debe a que, al hacer la llamada a `sleep` con tiempo 0, el procesador expulsa al proceso en cuestión, aunque vuelva a estar disponible inmediatamente y se encola antes que el otro proceso (por tanto, su ejecución se alterna).

Cabe destacar que al ejecutar el programa en Windows 10 (con el bash de Ubuntu que este ofrece), solo se producen escrituras. Esto puede deberse al uso de semáforos débiles que desbloquean al proceso padre primero por motivos desconocidos, a diferencia de la ejecución con Ubuntu (descrita arriba), que usa semáforos robustos.

- c) Si se establecen inicialmente 10 lectores y 1 segundo de espera, el resultado serán escrituras separadas por una serie de lecturas, ya que al poder producirse lecturas simultáneas, esto hará que se puedan ejecutar todos los procesos lectores a la vez. El proceso padre aprovechará el bloqueo de 1 segundo tras la lectura para volver encolarse en el semáforo y, eventualmente, volver a escribir.
- d) Al poner de nuevo el tiempo de espera en 0 segundos, el padre comenzará probablemente escribiendo, pero luego los 10 hijos comenzarán a leer simultáneamente y de forma continuada (el número de lectores se mantiene en un valor no nulo ya que los primeros lectores logran encolarse de nuevo antes de que los últimos liberen el recurso), lo que produce inanición al proceso escritor.
- e) La eliminación de la instrucción `sleep` hace que no sólo no haya espera sino que además el proceso no se llegue a bloquear. Esto produce que el proceso escritor pueda encadenar varias escrituras consecutivas al inicio de la ejecución (depende del tiempo que le sea asignado por el procesador), pero una vez comiencen a ejecutarse los procesos lectores, este sufrirá inanición de nuevo.