

# MEMORIA PRÁCTICA 2

CLAUDIA GONZÁLEZ ARTEAGA Y LAURA INIESTA RODRÍGUEZ

1. a)

```
e397697@8A-29-8-29: ~
Archivo Editar Ver Buscar Terminal Ayuda
e397697@8A-29-8-29:~$ man kill
e397697@8A-29-8-29:~$ kill --list
bash: kill: -list: especificación de señal no válida
e397697@8A-29-8-29:~$ kill -l
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
```

b) la señal SIGKILL tiene el número 9 y la señal SIGSTOP tiene el número 19.

```
e397697@8A-29-8-29:~$ kill -l sigkill
9
e397697@8A-29-8-29:~$ kill -l sigstop
19
e397697@8A-29-8-29:~$
```

2.

- a) Ejercicio\_kill.c adjunto.
- b) Cuando probamos el programa enviando la señal sigstop, si intentamos escribir en la terminal donde hemos introducido ps, no podemos(el proceso está muerto). Cuando le enviamos la señal sigcont, volvemos a la otra terminal y al escribir nos aparece en la terminal lo escrito anteriormente y lo que hemos escrito en ese momento.

3.

- a) Al llamar a sigaction le estamos pasando el manejador que va a usar en caso de recibir la señal por lo que el programa en cuanto reciba la señal indicada, en este caso SIGINT ejecutará el manejador.
- b) Si, se mata a la señal que el manejador está esperando por lo que el programa recibe la señal pero vuelve a pedirla, no se bloquea ninguna otra señal ya que se hace sigemptyset (&( act . sa\_mask ) ) ; lo que inicia la máscara a “empty” y eso implica que no tiene ninguna señal que se haya que bloquear guardada por lo que no se bloquea ninguna.
- c) Se imprime el printf “señal número %d recibida \n” cuando recibe la señal que está esperando, es decir, cuando sigaction recibe sigint(^c) pero vuelve a pedirlo debido al bloqueo de la señal y nunca llega a hacer el sleep.

4.
  - a) Cuando se reciba la señal sigaction puede tomar el valor SIGDFL que indica que la acción a realizar cuando se recibe la señal es la acción por defecto asociada a la señal (manejador por defecto). Esta acción suele ser terminar el proceso, y, en algunos casos, generar un fichero core.
  - b) No se pueden capturar todas las señales porque por definición hay dos señales que no pueden ser capturadas: sigkill(9) y sigstop(19).
5.
  - a) El nuevo if se encarga de manejar la señal ya que el manejador lo único que hace es activar la variable global got\_signal de forma que el if reconoce que se ha recibido la señal y actúa en consecuencia sacando el mensaje "Señal recibida" por pantalla por lo que es el encargado de gestionar la señal.
  - b) Como la llamada al manejador es asíncrona el tener una variable global permite acceder a ella en cualquiera que sea el momento del uso del manejador y permite al resto del código tener acceso a ella.
6.
  - a) Cuando le pasamos SIGUSR1 o SIGUSR2 no sucede nada porque esas señales no se encuentran en la máscara, por lo que son ignoradas. Sin embargo, cuando introducimos SIGINT, como éste si está en la máscara, finaliza el programa.
  - b) Cuando se termina la espera termina el programa y se escribe Fin del Programa porque, al hacer sleep, el proceso no sabe que se habían bloqueado ni SIGUSR1 ni SIGUSR2 por lo que al mandar SIGUSR1 el programa finaliza.
7.
  - a) Se llama al manejador, interrumpiendo la cuenta y se imprime por pantalla: "Estos son los números que me ha dado tiempo a contar".
  - b) Cuando comentamos la señal sigaction el manejador no se ejecuta ya que no le hemos asignado ninguno a la señal por lo que se ejecuta el manejador por defecto que finaliza el programa y ocurre lo siguiente :

```
e397697@8A-29-8-29:~/UnidadH/SOPER/Práctica2$ ./ej
Comienza la cuenta (PID=4953)
Temporizador
```

8.

- a) ejercicio\_prottemp.c adjunto.
- b) tras añadir el contador hemos podido ver que en cada ejecución del programa aunque tenga los mismos argumentos calcula un número distinto de señales, esto se debe a que no hay ningún control sobre la variable que comparten todos y como hay concurrencia, los procesos pueden acceder al sumador “a la vez”, es decir la variable puede empezar en 0 y dos hijos la usan a la vez luego para ambos vale 1 y aunque en realidad hay dos hijos con dos señales recibidas por la concurrencia la señal contabiliza 1 además, debido a la falta de control sobre los manejadores que también son compartidos por todos, su ejecución se superpone.

```
laura@laura-UX331UA:~/Desktop/UAM/SOPER/Practicas/Practica2$ ./ej8 5 10
El hijo con PID = 7629 suma: 290703.0
Se ha enviado sigusr2 al padre
El hijo con PID = 7630 suma: 291466.0
PID: 7629 TERMINADO
Se ha enviado sigusr2 al padre
PID: 7630 TERMINADO
El hijo con PID = 7632 suma: 291466.0
Se ha enviado sigusr2 al padre
PID: 7632 TERMINADO

2
El hijo con PID = 7631 suma: 291466.0
Se ha enviado sigusr2 al padre
PID: 7631 TERMINADO

1
El hijo con PID = 7634 suma: 291466.0
Se ha enviado sigusr2 al padre
PID: 7634 TERMINADO

3
4
Se han recibido 4 señales
```

En la imagen podemos ver cómo se crean 5 hijos pero solo se contabilizan 4 señales.

9.

Si, se puede modificar su posición ya que `sem_unlink` no elimina de inmediato el semáforo, sólo su nombre y espera a que todos los procesos que lo han abierto lo cierren para eliminarse.

Se podría poner justo debajo del `sem_open`, `sem_unlink` una vez que el padre y el hijo cierran el semáforo y el resto de procesos que tenga asociados borrará el semáforo por completo.

No se puede poner antes de `sem_open` ya que elimina el nombre del semáforo y se crearía uno nuevo al hacer el `sem_open`.

El hijo que se crea con el `fork()` tiene acceso al semáforo ya abierto por lo que no necesita volverlo a abrir.

10.

- a) En cuanto se recibe SIGINT se imprime por pantalla el mensaje: Fin de la espera. El semáforo está en wait delante de ese mensaje, esperando a que se libere con un sem\_post para poder continuar, la llegada de esta señal hace que el semáforo despierte pero devuelve un error, cómo no hay control de errores en sem\_wait se continúa con la ejecución del código.
- b) Al ignorar la señal el programa se queda bloqueado ya que se está esperando al semáforo que en ningún momento ha sido liberado, al introducir la señal SIG\_INT no sucede nada.

```
laura@laura-UX331UA:~/Desktop/UAM/SOPER/Practicas/Practica2$ ./ej
Entrando en espera (PID=23283)
^C^C^C^C^C^C
```

- c) Para ello habría que realizar un control de errores en sem\_wait, comprobando si el valor es EINTR (fallo por una señal) o no de forma que el programa solo se ejecutará por completo si se libera el semáforo.

11.

Liberamos el semáforo uno justo antes de solicitarlo en la primera posición a imprimir (1) y lo liberamos después de usarlo, el 3 espera al otro semáforo que todavía no ha sido liberado por lo que está bloqueado, el 2 espera al 1 que queda libre justo tras ejecutar el 1 por lo que se imprime 1 y luego 2, éste libera el semáforo 2 de forma que 3 lo puede usar pero no libera el semáforo 1 porque 4 le está esperando y de esta forma queda bloqueado a la espera de que el semáforo 1 se libere, lo liberamos tras ejecutar 3 de forma que obtenemos la siguiente salida:

```
laura@laura-UX331UA:~/Desktop/UAM/SOPER/Practicas/Practica2$ ./ej
1
2
3
4
```

Código ejercicio\_alternar.c adjunto.

12. Código ejercicio\_prottemp\_mejorado.c adjunto

14. a) Código ejercicio\_lect\_escr.c adjunto

b) Para SECS = 0 y NREADS = 1 solo escribe

```
claudia@Claudia
W-INI <11782>
W-fln <11782>
R-INI <11783>
R-fln <11783>
W-INI <11782>
W-fln <11782>
R-INI <11783>
R-fln <11783>
W-INI <11782>
W-fln <11782>
R-INI <11783>
R-fln <11783>
W-INI <11782>
W-fln <11782>
```

c) Para SECS = 1 y NREADS = 10 se producen lecturas hasta la finalización del programa. De la misma manera, el escritor debe estar escribiendo hasta fin del programa, es decir, se producen escrituras hasta el final del programa.

```
claudia@Claudia:~/t
R-INI <11644>
R-INI <11645>
R-INI <11647>
R-INI <11646>
R-INI <11648>
R-INI <11649>
R-INI <11650>
R-INI <11651>
R-INI <11652>
R-INI <11653>
R-fln <11644>
R-fln <11645>
R-fln <11648>
R-fln <11646>
R-fln <11647>
R-fln <11649>
R-fln <11650>
```

d) Para SECS = 0 y NREADS = 10 se producen sólo lectura.

```
claudia@Claudia:~
R-INI <12024>
R-INI <12025>
R-INI <12026>
R-INI <12027>
R-INI <12028>
R-INI <12029>
R-INI <12030>
R-INI <12031>
R-INI <12032>
R-INI <12033>
R-fln <12024>
R-INI <12024>
R-fln <12026>
R-INI <12026>
R-fln <12025>
R-fln <12030>
R-fln <12027>
```

e) Si se elimina totalmente el sleep del bucle no se produce ninguna lectura.

[illegible]