

PRÁCTICA 2

SEÑALES Y

SEMÁFOROS

Ejercicio 1:

- a) Usando el comando “kill -l” o “kill -L” podemos obtener la lista de señales

```
eps@labvirtpeps: ~/Escritorio/uni/SOPER/P2
Archivo Editar Ver Buscar Terminal Ayuda
-SIGKILL or -KILL. Negative PID values may be used to choose whole
process groups; see the PGID column in ps command output. A PID of -1
is special; it indicates all processes except the kill process itself
and init.

OPTIONS
<pid> [...]
    Send signal to every <pid> listed.

-<signal>
-s <signal>
--signal <signal>
    Specify the signal to be sent. The signal can be specified by
    using name or number. The behavior of signals is explained in
    signal(7) manual page.

-l, --list [signal]
    List signal names. This option has optional argument, which
    will convert signal number to signal name, or other way round.

-L, --table
    List signal names in a nice table.

Manual page kill(1) line 13 (press h for help or q to quit)
```

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

```
eps@labvirtpeps: ~/Escritorio/uni/SOPER/P2$ 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

eps@labvirtpeps:~/Escritorio/uni/SOPER/P2$ kill -l | grep "SIGKILL"
6) SIGABRT      7) SIGBUS      8) SIGFPE       9) SIGKILL     10) SIGUSR1
eps@labvirtpeps:~/Escritorio/uni/SOPER/P2$ kill -l | grep "SIGSTOP"
16) SIGSTKFLT  17) SIGCHLD   18) SIGCONT     19) SIGSTOP    20) SIGTSTP
eps@labvirtpeps:~/Escritorio/uni/SOPER/P2$
```

Ejercicio 2:

- a) El código que aparece a continuación es el trozo que hay que incluir en el programa para reproducir de forma limitada la funcionalidad del comando “kill”

```
if((kill(pid, sig))){
    perror("Kill");
}
```

- b) Puesto que se suspende el proceso tras recibir SIGSTOP, no se puede escribir nada en la terminal, sin embargo, cuando recibe SIGCONT, aparecen todos los mensajes escritos mientras se encontraba la terminal parada.

Ejercicio 3:

- a) Sí, al llamar a “sigaction” se ejecuta consigo el manejador. Si no se recibe la señal indicada no se ejecutará ninguno de los dos.
- b) Durante la ejecución del manejador no se bloquea ninguna señal, dado que la máscara está vacía.
- c) El “printf” aparece cuando se ejecuta el planificador, es decir, cuando se recibe la señal esperada en “sigaction”.
- d) Al modificar el programa y no capturar SIGINT, cuando se reciba la señal, se llama al manejador por defecto, que, por lo general, termina el proceso. Incluso, puede generar un fichero “core”.
- e) No se pueden capturar todas las señales, porque, como dice el manual, la señales SIGKILL (9) y SIGSTOP (19) no pueden ser capturadas ni ignoradas.

Ejercicio 4:

- a) La gestión de la señal se realiza al entrar en if de la línea de código “if(got_signal)”, la gestión de la señal corresponde a reiniciar el flag “got_signal” a 0 e imprimir por pantalla que se ha recibido correctamente.
- b) Se permite el uso de variables globales para que, al salir del manejador, el proceso tenga el valor del flag “got_signal” cambiado y pueda tratar la señal, que estaba bloqueada.

Ejercicio 5:

- a) Al recibir las señales SIGUSR1 y SIGUSR2, el programa no reacciona ya que estas señales están bloqueadas por la máscara, es decir, las deja pendientes y bloqueadas. Sin embargo, al recibir SIGINT, la señal no se bloquea y, por lo que está establecido, tampoco se ignora, por lo que se lleva a cabo la rutina por defecto, finalizando la ejecución.

```
eps@labvirteps:~/Escritorio/uni/SOPER/P2/E5$ ./sig_sigset
En espera de señales (PID = 8055)
SIGUSR1 y SIGUSR2 están bloqueadas
Terminado (killed)
eps@labvirteps:~/Escritorio/uni/SOPER/P2/E5$ █
```

```
eps@labvirteps:~$ kill -10 8055
eps@labvirteps:~$ kill -12 8055
eps@labvirteps:~$ kill -9 8055
eps@labvirteps:~$ █
```

b) A continuación se muestran los cambios realizados en el código para llamar a “sleep” en lugar de a “pause” y la salida obtenida.

```
sleep(10); /*espera 10 segundos*/  
if (sigprocmask(SIG_UNBLOCK, &set, &oset) < 0) {  
    perror("sigprocmask");  
    exit(EXIT_FAILURE);  
}  
printf("Máscara original\n");  
sleep(10); /*espera 10 segundos*/
```

```
eps@labvirteps:~/Escritorio/uni/SOPER/P2/E5$ ./sig_sigset  
En espera de señales (PID = 9072)  
SIGUSR1 y SIGUSR2 están bloqueadas  
Máscara original  
Señal definida por el usuario 1  
eps@labvirteps:~/Escritorio/uni/SOPER/P2/E5$   
eps@labvirteps:~$ kill -10 9072  
eps@labvirteps:~$
```

Al finalizar la espera se recibe el mensaje “Mascara original” y “Señal definida por el usuario 1”. No se imprime el mensaje de despedida, porque, al haber restaurado la máscara a la original, la señal de SIGUSR1, que estaba bloqueada antes del “sleep”, deja de estarlo y pasa a ser tratada por el manejador por defecto, el cual termina la ejecución y por eso no se recibe el mensaje de despedida.

Ejercicio 6:

- Si se envía la señal SIGALARM al proceso mientras se realiza la cuenta, este termina llamando al manejador y se escribe por pantalla el mensaje del manejador junto con el número máximo alcanzado.
- Si comentamos la llamada a “sigaction”, es como si la eliminamos y, por tanto, el proceso termina sin llamar al manejador y aparece por pantalla el mensaje “terminado”.

Ejercicio 7:

También se podría colocar el “sem_unlink” antes de finalizar el proceso hijo, puesto que después no será necesario el uso del semáforo.

```
if (pid == 0) {
    sem_wait(sem);
    printf("Zona protegida (hijo)\n");
    sleep(2);
    printf("Fin zona protegida (hijo)\n");
    sem_post(sem);
    sem_close(sem);
    sem_unlink(SEM_NAME);
    exit(EXIT_SUCCESS);
}
else {
    sem_wait(sem);
    printf("Zona protegida (padre)\n");
    sleep(2);
    printf("Fin zona protegida (padre)\n");
    sem_post(sem);
    sem_close(sem);

    wait(NULL);
    exit(EXIT_SUCCESS);
}
```

Ejercicio 8:

- a) Cuando se envía la señal “SIGINT”, el proceso termina llamando al manejador. El “sem_wait” se realiza correctamente esperando al SIGINT puesto que, al recibirlo, el semáforo se abre y continua la ejecución hasta el mensaje de “Fin de espera” y la finalización de la ejecución.
- b) Si, en lugar del manejador vacío, se ignora la señal con SIG_IGN, el proceso no termina.
- c) Para que el programa espere a que el semáforo haga “down”, lleguen o no señales, hay que crear un bucle cuya condición sea la bajada del semáforo.

Ejercicio 9:

Para este ejercicio tenemos que sustituir los siguientes códigos:

Código A: `sem_wait(sem2);`

Código B: `sem_post(sem1); sem_wait(sem2);`

Código C: `sem_post(sem1);`

Código D: `sem_post(sem2); sem_wait(sem1);`

Código E: `sem_post(sem2); sem_wait(sem1);`

Ejercicio 10:

f) No se pueden hacer suposiciones sobre la velocidad relativa de los procesos, ya que depende de factores externos como planificador, política de interrupciones y de otros procesos, por lo que, sin que haya una sincronización entre todos, no se puede determinar el orden en el que se imprimen.