PRÁCTICA 2
SEÑALES Y SEMÁFOROS

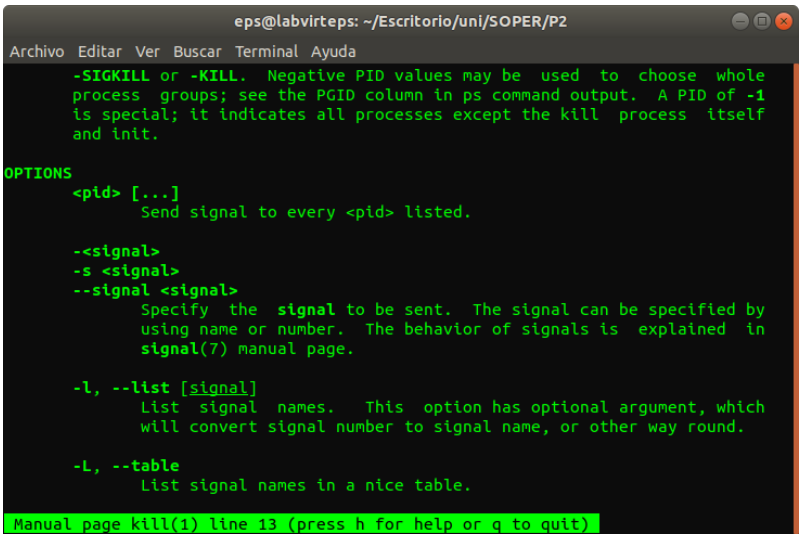
Por Daniel Cerrato y David Garitagoitia

2º Curso – Sistemas Operativos, Prácticas

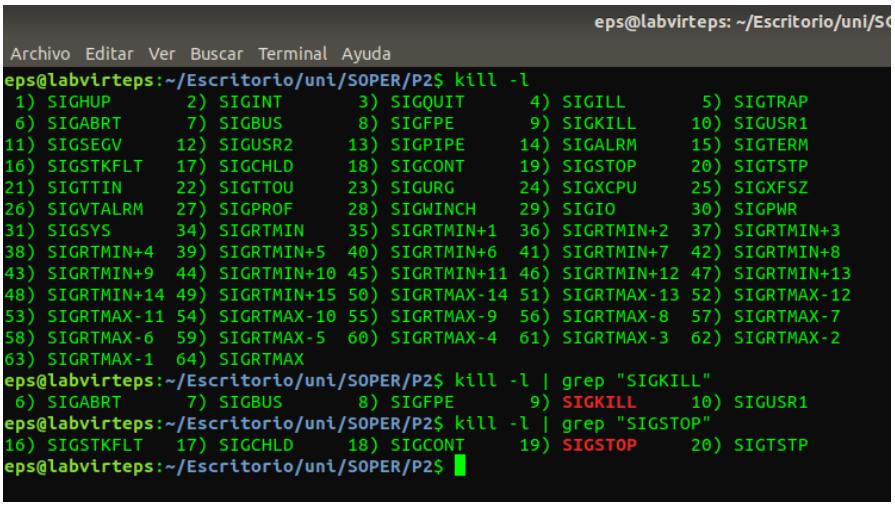
Escuela Politécnica Superior, Universidad Autónoma de Madrid

Ejercicio 1:

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



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



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.

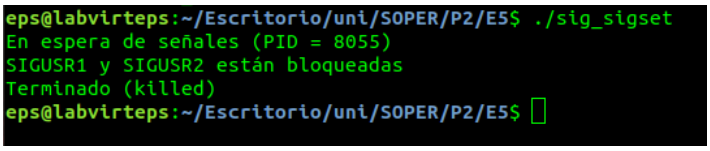
Ejercicio 4:

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.



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:

a) 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.

b) 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.