Miguel Arnáiz Montes
S. Xiao Fernández Marín
G2291_P07

PRÁCTICA 2

Ejercicio 1

a) kill -l

b) SIGKILL 9
SIGSTOP 19

Ejercicio 2

a) Archivo adjunto (ejercicio_kill.c)

b) Si se ejecuta SIGSTOP en una terminal, la segunda terminal, a la que mandas la señal no puedes seguir escribiendo.
Si luego le envías la señal SIGCONT (18) se vuelve a poder escribir.

Ejercicio 3

a) No. Solamente indica que se ejecutará el manejador en caso de recibir SIGINT.

b) Se ejecutarán las señales que lleguen ya que la máscara de señales que utilizamos está vacía: `sigemptyset(&(act.sa_mask));`

c) El printf aparece al llegar al while y cada vez que pulsemos Ctrl+C ya que estamos en un bucle infinito. Para terminar el programa utilizaremos Ctrl+Z.

Ejercicio 4

a) Se ejecuta la rutina del sistema por defecto.

b) No se pueden capturar, ni ignorar, ni bloquear las señales SIGKILL(9) y SIGSTOP(19). Esto es debido a que estas señales dan orden de matar o parar el proceso actual, si estas señales se pudieran ignorar y el proceso entrase en un bucle infinito habría manera de finalizarlo.

Ejercicio 5

a) La gestión de la señal se realizan en el manejador y en

```
if (got_signal){
       got_signal = 0;
       printf ("Señal recibida. \n");
}
```

b) Esto es debido a que el manejador tiene que ser del tipo ***void manejador(int sign)*** no existe un espacio en el que reflejar que se ha recibido una señal en el main. Por esta razón se utiliza una variable global para poder comunicarse entre el manejador y el main.

Miguel Arnáiz Montes
S. Xiao Fernández Marín
G2291_P07

Ejercicio 6

a) Las señales SIGUSR1 y SIGUSR2 se meten en una cola mientras estén bloqueadas, hasta que el programa termine, (que no es el caso ya que nos encontramos con un pause()) donde las desbloqueará y el sistema operativo terminará ejecutándolas.
Cuando se recibe SIGINT la terminal recibe la señal, ya que no está bloqueada, y termina con el programa.

b) Cuando finaliza la espera el programa recibe lo que tenía en la cola ya se desbloquea las señales mandadas a la ocla, utilizando SIG_UNBLOCK.

Ejercicio 7

a) Se ejecutará la función manejador_SIGALARM establecida previamente a iniciar la cuenta como manejador de la señal SIGALARM. En esta función se ejecuta un printf y provoca que se salga del programa, dejando de contar.

b) Al recibir SIGALARM se muestra en la pantalla: “Temporizador” y sale del programa.

Ejercicio 8

a) Archivo adjunto (ejercicio_prottemp.c). Comentado en el código.

b) Siempre se van a recibir menos señales menos señales SIGUSR2 que hijos creados (N). Esto es debido a que cada hijo puede mandar como mucho tan solo 1 señal, aunque esto no quiere decir que la llegue a mandar ya que puede recibir la señal SIGTERM del padre antes. No hay manera de saber cuantas SIGUSR2 va a recibir el padre debido a la velocidad relativa de los procesos.

Ejercicio 9

Sí. Se podría poner también sem_unlink tras el uso que le da el hijo, para que al llegar el contador de procesos asociados al semáforo a cero, sus recursos sean liberados por el sistema operativo.

Ejercicio 10

a) Al recibir la señal SIGINT el programa termina. La llamada a sem_wait no se ejecuta con éxito, ya que como ya está a 0, lo que hace es bloquearlo porque no puede restarle 1.

b) No se consigue salir del programa utilizando ctrl+c.

c) Bloqueamos todas las señales para que no interfieran en el down del semáforo.

Ejercicio 11

```
        ...
if (pid == 0){
```

```
        /* Rellenar Código A */
        printf("1\n");
        /* Rellenar Código B */
        sem_post(sem1);
        sem_wait(sem2);
        sem_post(sem1);
        printf("3\n");
        /* Rellenar Código C */
        sem_close(sem1);
        sem_close(sem2);
} else if (pid > 0){
        /* Rellenar Código D */
        sem_wait(sem1);
        printf("2\n");
        sem_post(sem2);
        /* Rellenar Código E */
        sem_wait(sem1);
        printf("4\n");
        /* Rellenar Código F */
        sem_close(sem1);
        sem_close(sem2);
                ...
}
```

Ejercicio 12

Archivo adjunto (ejercicio_prottemp_mejorado.c)

Ejercicio 14

a) Archivo adjunto (ejercicio_lect_escr.c)

b) Se producen escritura y lectura correspondiente al padre e hijo.

c) Se producen los 10 reads y tras esto el write, ya que hasta que hast que el read no se haya impreso entero, no se abre el semáforo del write.

d) Se producen solo los reads, ya que al no esperar tras haber ejecutado el código del hijo, al semáforo que controla el escritor, no le da tiempo a ejecutarse.

e) Se producen tan solo los reads, pero se imprime en orden, es decir, en los apartados anteriores se imprimían los 10 reads en orden y tras esto, se volvía a repetir.