Sistemas Operativos
Práctica 2

Antonio Javier Casado Hernández

Escuela Politécnica Superior

Semana 1

Ejercicio 1

a) En el manual (`man kill`) se especifica especifica -l para mostrar la lista: `kill -l`

b) `SIGKILL` tiene número 9 y `SIGSTOP` tiene número 19

Ejercicio 2

a) Código ejercicio_kill.c, se ha utlizado la función `kill()` de C.

b) `SIGSTOP` para el proceso, no se pude escribir hasta que se le hace `SIGCONT`

Ejercicio 3

a) No, la llamada a *sigaction* arma el manejador, este será ejecutado cuando reciba una señal `SIGINT`

b) Se bloquea `SIGINT` para no volver a llamar al manejador si llegan mas SIGINTs.

c) Aparece cuando se le manda la señal CTRL + C.

Ejercicio 4

a) Cuando un programa recibe señal y no tiene manejador termina.

b) Según el manual, se puede capturar todas las señales excepto `SIGKILL` y `SIGACTION` usando un bucle.

Ejercicio 5

a) La gestión se realiza Dentro del *if* en el main.

b) Se usa variables globales para que el manejador pueda proporcionar algún tipo de información para el programa principal.

Ejercicio 6

a) Cuando recibe `SIGUSR1` o `SIGUSR2` no ocurre nada porque esas dos señales están bloqueadas. Con SIGINT se cierra

b Cuando finaliza la espera el programa termina por la señal `SIGUSR1`, no se imprime porque al haberse establecido la mascara anterior (que está vacía), no bloquea la señal.

Ejercicio 7

a) Cuando se envía una señal se activa el manejador y se termina el programa.

b) Si se comenta *sigaction* el programa se cierra sin llamar al manejador.

Ejercicio 8

a) Código ejercicio_prottemp.c Las decisiones de diseño para la suspensión del proceso padre han sido que dentro de un bucle se compruebe si la variable global *got_signal_alrm* es distinta de 0, cuando la señal llega, la llamada a `sleep` es cancelada volviendo al bucle y entrando en la condición en la que hay señal de alarma, aquí dentro se envía señal de `SIGTERM` a todos los hijos.
Igualmente en los hijos se comprueba en el bucle en el main si se ha recibido la señal `SIGTERM` del padre, de esta forma, cuando se reciba la señal, los hijos salen del bucle pudiendo liberar recursos y terminar.

Se ha modularizado las funciones de este ejercicio en los archivos *prottemp.c* y su cabecera *prottemp.h*.

b) En ocasiones el contador de señales recibidas son menos que los procesos. No se puede garantizar que se registren todas las señales debido a que los procesos puede que envíen señales al mientras se está ejecutando el manejador de producido por otro proceso, esto hará que el manejador no sea ejecutado para el que llegó último.

Ejercicio 9

La primera posición en la que se podría poner `sem_unlink` sería después de hacer `sem_wait` en el padre.

Ejercicio 10

a) Cuando llega `SIGINT` el proceso puede "pasar" el semáforo. Porque la señal termina la llamada a la función.

b) Usando `SIG_IGN` ignora la señal y el proceso sigue en espera del semáforo.

c) Habría que poner `sem_wait` dentro de un bucle, si se interrumpe se establece `errno` a EINTR. Mientras el retorno de `sem_wait` sea -1 y `errno` sea EINTR, se ejecuta el bucle, i.e:

```
1       while(sem_wait(sem) == -1 && errno == EINTR)
2           continue;
```

Ejercicio 11

```
1           /*---HIJO---*/
2           /* Rellenar Codigo A */
3           printf("1\n");
4
5           /* Rellenar Codigo B */
6           sem_post(sem1);
7           sem_wait(sem2);
8           printf("3\n");
9
10          /* Rellenar Codigo C */
11          sem_post(sem1);
12
13          /*---PADRE---*/
14          /* Rellenar Codigo D */
15          sem_wait(sem1);
16          printf("2\n");
17
18          /* Rellenar Codigo E */
19          sem_post(sem2);
20          sem_wait(sem1);
21          printf("4\n");
22
23          /* Rellenar Codigo F */
```

Ejercicio 12

a) Para el código de *ejercicio_prottemp_mejorado.c* se ha utilizado una versión del problema *lectores-escritores*:

En primer lugar los procesos hijos, cuando acaban su operación leen y escriben de uno en uno el archivo *data.txt*, de esta forma se evita que dos o más procesos lean y actualicen incorrectamente el archivo (por ejemplo, terminan dos procesos pero ambos leen 0 en la linea de procesos terminados por lo que terminan escribiendo que ha terminado 1).

El padre sí usa la parte del algoritmo de lectores para leer el archivo.

Se ha modularizado el programa en los archivos *prottemp_ mejorado.c* para las implementaciones de las funciones junto con su cabecera *prottemp_ mejorado.h* y el main en el archivo *ejercicio_ prottemp_ mejorado.c*

Ejercicio 14

a) Se ha implementado el algoritmo de lectores-escritores, en los bucles (de los lectores y de los escritores), se comprueba si alguna señal ha llegado, la "gestión" de la señal se hace en el propio *main* (no dentro del handler) debido a que se debe liberar memoria en los hijos y padre y enviar señal del padre a los hijos.

b) Sí se producen ambas lecturas y escrituras porque debido a que es un solo proceso lector, cuando este salga de la lectura hará un UP del semáforo para el escritor haciendo que este pueda escribir.

c) Sí se producen ambas lecturas y escrituras porque el `sleep` hace que de tiempo a todos los lectores a acabar de leer y el último podra hacer UP del escritor

d) Solo se producen lecturas porque, al no hacer `sleep` los procesos lectores y estos teniendo prioridad, siguen entrando de nuevo (mientras hay otros leyendo también) simulando un número "infinito" de lectores y dando la imposibilidad de hacer UP del semaforo para el escritor.

e) Sí se producen ambas lecturas y escrituras porque sin ningún `sleep`, se da la posibilidad de que todos los procesos lectores salgan de la zona de lectura, haciendo así que el escritor entre.