Ejercicio 3.3 - Señales

In [ ]:

```
%%bash
rm -rf *.c *.o *.elf Makefile

/*
Daniel Ledesma Ventura
Badr Guaitoune Akdi
*/
```

Ejercicios

Ejercicio 1 - handler.c Escribe un programa que capture las señales SIGINT y SIGTSTP, para que se incremente un contador independiente ante cada recepción de dichas señales. El programa entrará en un bucle hasta que se hayan recibido un total de 10 señales, en cuyo caso mostrará por pantalla el número de señales de cada tipo recibidas, y finalizará. Desde otro terminal, envía señales usando el comando `kill`. Usa también las combinaciones de teclado adecuadas para enviar dichas señales desde el terminal en el que se ejecuta el proceso. ¿Qué combinación de teclas envía cada señal desde el teclado?

Respuesta:

SIGINT (^ C), SIGTSTP (^ Z) y SIGQUIT (^).La ultima ^ \ para enviar SIGQUIT, actúa como un error y provoca un volcado de núcleo de forma predeterminada, si se laza provoca (`core' generado). Linea de comandos 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

...

In [ ]:

```
%%writefile handler.c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

static int contador_STP=0;
static int contador_INT=0;

void handler(int signo) {
  char m1[80]="Recibida señal SIGTSTP\n";
  char m2[80]="Recibida señal SIGINT\n";

  if (signo == SIGTSTP) {
    write(1, m1, 80 );
    contador_STP++;
  }

  if (signo == SIGINT) {
    write(1, m2, 80 );
    contador_INT++;
  }
}

int main() {

  struct sigaction act;

/*
```

```
1. Inicializamos el puntero de la funcion
*/
/*************************************************************/
     act.sa_handler = handler;
/*************************************************************/

/*
2. Creamos el conjunto de señales en act con SIGINT y SIGSTP
*/
/*************************************************************/
     sigemptyset(&act.sa_mask);
     sigaddset(&act.sa_mask, SIGINT);
     sigaddset(&act.sa_mask, SIGTSTP);
/*************************************************************/

/*
3. Opciones del handler a SA_RESTART
*/
/*************************************************************/
     act.sa_flags = SA_RESTART;
/*************************************************************/

/*
4. Instalamos el controlador para las dos señales
*/
/*************************************************************/
     sigaction(SIGINT,  &act, NULL);
     sigaction(SIGTSTP, &act, NULL);
/*************************************************************/

  while (contador_STP+contador_INT<10) {};
  printf("Se recibieron %d señales: %d SIGINT y %d SIGTSTP\n", contador_STP+contador_INT, contador
_INT,contador_STP);
}

/*

badr@badr:~/Escritorio/ASO/Ejercicio3/Ejercicio3.3$ ./handler.elf
^ZRecibida señal SIGTSTP
^ZRecibida señal SIGTSTP
^CRecibida señal SIGINT
^CRecibida señal SIGINT
^CRecibida señal SIGINT
^CRecibida señal SIGINT
^CRecibida señal SIGINT
^ZRecibida señal SIGTSTP
^ZRecibida señal SIGTSTP
^ZRecibida señal SIGTSTP
Se recibieron 10 señales: 5 SIGINT y 5 SIGTSTP


badr@badr:~/Escritorio/ASO/Ejercicio3/Ejercicio3.3$ ./handler.elf
^\Abandona (`core' generado)


*/
```

In [ ]:

```
%%bash
gcc handler.c -o handler.elf
# Este ejercicio hay que probarlo en terminal
# usuarioso@fb6aefc36de3:~/aso-jupyter-public/ejercicios/ejercicio3.2$ ./handler.elf
# Recibida señal SIGINT
# Recibida señal SIGINT
# Recibida señal SIGINT
# Recibida señal SIGINT
# Recibida señal SIGINT
# Recibida señal SIGINT
# Recibida señal SIGTSTP
# Recibida señal SIGTSTP
# Recibida señal SIGTSTP
# Recibida señal SIGTSTP
# Se recibieron 10 señales: 6 SIGINT y 4 SIGTSTP
```

Ejercicio 2 - suspender.c Escribe un programa que se suspenda hasta la recepción de una señal de tipo SIGINT. En ese caso, el manejador imprimirá un mensaje indicando la recepción, y el programa finalizará.

1. ¿Qué llamada se utiliza para indicar el manejador de una señal?
2. ¿Y para expresar que se desea ignorar una señal?

Respuesta:

---

1. ... SIGACTION (sig, *act,* oact) Es una nueva versión de signal, examina, pone o modifica los atributos de una señal. Sig es un entero que indica la señal. *act es una estructura que contiene los atributos y manejador de la señal* oact es la estructura que recibe los atributos antes de la llamada.
2. ... Para cada tipo de señal el proceso puede especificar una acción diferente: SIG_IGN Ignorar la señal, lo que no es posible para todos los tipos de señales. Estructura sigaction De la estructura sigaction(*act), que describe la acción para una señal, los campos más relevantes son: sa_handler->Describe el manejador de la señal. Igual que ocurre con signal(), este campo puede valer SIG_DFL, SIG_IGN(para ignorar) o un puntero a una función.

In [ ]:

```
%%writefile suspender.c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

void handler(int signo) {
  char m[80]="Recibida señal SIGINT\n";
  write(1, m, 80 );
}

int main() {

  struct sigaction act;
  sigset_t seniales;

/*
1. Inicializamos el puntero de la funcion
*/
/*************************************************************/
   act.sa_handler = handler;
/*************************************************************/

/*
2. Creamos el conjunto de seniales en act a vacio
*/
/*************************************************************/
   sigemptyset(&act.sa_mask);
/*************************************************************/

/*
3. Opciones del handler a SA_RESTART
*/
/*************************************************************/
    act.sa_flags = SA_RESTART;
/*************************************************************/

/*
4. Instalamos el controlador para la señal SIGINT
*/
/*************************************************************/
    sigaction(SIGINT,  &act, NULL);
/*************************************************************/

/*
5. Crear el conjunto de señales por las que esperaremos
*/
/*************************************************************/
   sigfillset(&seniales);
   sigdelset(&seniales,SIGINT);
/*************************************************************/

/*
6. Suspender el proceso.
*/
/*************************************************************/
```

```
   sigsuspend(&seniales);
/*****************************u*****************************/

  printf("Continuo la ejecucion\n");
  return 1;
}


/*
badr@badr:~/Escritorio/ASO/Ejercicio3/Ejercicio3.3$ ./suspender.elf
^CRecibida señal SIGINT
Continuo la ejecucion
*/
```

In [ ]:

```
%%bash
gcc suspender.c -o suspender.elf
# Este ejercicio hay que probarlo en terminal
# usuarioso@fb6aefc36de3:~/aso-jupyter-public/ejercicios/ejercicio3.2$ ./suspender.elf
# ^CRecibida señal SIGINT
# Continúo la ejecución
```

Ejercicio 3 - killer.c Implementa un programa que, a modo de terminal, reciba por entrada estándar un identificador de proceso. Una vez recibido, el programa enviará una señal SIGKILL a dicho proceso, controlando los posibles errores que puedan surgir e informando al usuario de los mismos. Utilízalo para finalizar el proceso creado en el primer apartado (`handler`). Posteriormente, prueba a crear el proceso *handler* con otro usuario (*osuser* por ejemplo) y vuelve a usar *killer* para enviar señales (con usuario *usuarioso*).

1. ¿Qué ocurre?
2. ¿Qué llamada al sistema empleas para enviar una señal a un proceso?

Respuesta:

---

1. ... Nada se mata el proceso no he podidio realizar la prueba con otro usuario.
2. ... kill -15 1234 llamada opcion(listada en el ejercicio 1) pid->proceso

In [ ]:

```
%%writefile killer.c
/*
Includes:
*/
/*****************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/types.h>
#include <errno.h>
 #include <unistd.h>
/*****************************************************************/

#define PIDSIZE  6

extern int errno;

int main() {
  int pid;
  char c_pid[7];
  c_pid[6]='\0';

  // Leer el pid de la entrada estandar
  fgets(c_pid, PIDSIZE, stdin);

  while (strcmp(c_pid,"quit\n")!=0) {
    if (strcmp(c_pid,"ps\n")==0) {
/*
Ejecutar la orden ps
*/
/*****************************************************************/
```

```
     system("ps");
/*************************************************************/
    } else {
      pid=atoi(c_pid);
/*
Evitamos que se envie una señal a nuestro proceso
*/
/*************************************************************/
      if (pid == getpid())
/*************************************************************/
        printf("A mi no \n");
      else
/*
Envio efectivo de la señal, al proceso mensaje selectivo de error
*/
/*************************************************************/
        if (kill(pid,SIGKILL) == -1) {
/*************************************************************/
          printf("\t no puede matarlo!!! ");
          if (errno == ESRCH)
            printf("(no existe)\n");
          if (errno == EPERM)
            printf("(no se deja)\n");
        }
        else
          printf("\t muerto!!! \n");
    }
    // Nuevo pid
    fgets(c_pid, PIDSIZE, stdin);
  }
  return 0;
}


/*
badr@badr:~/Escritorio/ASO/Ejercicio3/Ejercicio3.3$ nano &
[5] 8486
badr@badr:~/Escritorio/ASO/Ejercicio3/Ejercicio3.3$ ./killer.elf
ps
  PID TTY          TIME CMD
 6037 pts/2    00:00:00 bash
 6815 pts/2    00:00:00 suspender.elf
 7789 pts/2    00:00:00 nano
 8012 pts/2    00:00:00 nano
 8414 pts/2    00:00:00 nano
 8486 pts/2    00:00:00 nano
 8488 pts/2    00:00:00 killer.elf
 8490 pts/2    00:00:00 sh
 8491 pts/2    00:00:00 ps
208
  no puede matarlo!!! (no se deja)
ps
  PID TTY          TIME CMD
 6037 pts/2    00:00:00 bash
 6815 pts/2    00:00:00 suspender.elf
 7789 pts/2    00:00:00 nano
 8012 pts/2    00:00:00 nano
 8414 pts/2    00:00:00 nano
 8486 pts/2    00:00:00 nano
 8488 pts/2    00:00:00 killer.elf
 8506 pts/2    00:00:00 sh
 8507 pts/2    00:00:00 ps
8012
  muerto!!!
ps
  PID TTY          TIME CMD
 6037 pts/2    00:00:00 bash
 6815 pts/2    00:00:00 suspender.elf
 7789 pts/2    00:00:00 nano
 8414 pts/2    00:00:00 nano
 8486 pts/2    00:00:00 nano
 8488 pts/2    00:00:00 killer.elf
 8509 pts/2    00:00:00 sh
 8510 pts/2    00:00:00 ps
7789
  muerto!!!
q
```

```
[2]   Terminado (killed)      nano
[3]   Terminado (killed)      nano

[5]+  Detenido                nano
Terminado (killed)

--------------------------------------------------------------------------------

badr@badr:~/Escritorio/ASO/Ejercicio3/Ejercicio3.3$ ./handler.elf &
[6] 8611
badr@badr:~/Escritorio/ASO/Ejercicio3/Ejercicio3.3$ ./killer.elf
ps
  PID TTY          TIME CMD
 6037 pts/2    00:00:00 bash
 6815 pts/2    00:00:00 suspender.elf
 8414 pts/2    00:00:00 nano
 8486 pts/2    00:00:00 nano
 8611 pts/2    00:00:09 handler.elf
 8613 pts/2    00:00:00 killer.elf
 8614 pts/2    00:00:00 sh
 8615 pts/2    00:00:00 ps
8611
  muerto!!!
ps
  PID TTY          TIME CMD
 6037 pts/2    00:00:00 bash
 6815 pts/2    00:00:00 suspender.elf
 8414 pts/2    00:00:00 nano
 8486 pts/2    00:00:00 nano
 8613 pts/2    00:00:00 killer.elf
 8616 pts/2    00:00:00 sh
 8617 pts/2    00:00:00 ps
^C
[6]   Terminado (killed)      ./handler.elf



*/
```

In [ ]:

```
%%bash
gcc killer.c -o killer.elf
# Este ejercicio hay que probarlo en terminal, un ejemplo:
# usuarioso@fb6aefc36de3:~/aso-jupyter-public/ejercicios/ejercicio3.2$ nano &
# [1] 203
# usuarioso@fb6aefc36de3:~/aso-jupyter-public/ejercicios/ejercicio3.2$ ./killer.elf
# ps
#   PID TTY          TIME CMD
#   164 pts/2    00:00:00 sh
#   165 pts/2    00:00:00 bash
#   203 pts/2    00:00:00 nano
#   204 pts/2    00:00:00 killer.elf
#   205 pts/2    00:00:00 sh
#   206 pts/2    00:00:00 ps
# 208
#          no puede matarlo!!! (no existe)
# ps
#   PID TTY          TIME CMD
#   164 pts/2    00:00:00 sh
#   165 pts/2    00:00:00 bash
#   203 pts/2    00:00:00 nano
#   204 pts/2    00:00:00 killer.elf
#   207 pts/2    00:00:00 sh
#   208 pts/2    00:00:00 ps
# 203
#          muerto!!!
# quit
# [1]+  Killed                  nano
```

Ejercicio 4 - alarma.c Completa el programa de modo que se envíen señales de alarma al proceso hasta que finalice su ejecución.

In [ ]:

```
%%writefile alarma.c
/*
Includes:
*/
/*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/time.h>
#include <unistd.h>
#include <unistd.h>

/*****************************************************************/

volatile int contador=0;

void alarma (int signo) {
  char m[80]="Se acabo la cuenta\n";
  write(1,m,80);
  contador ++;
}

void* ini_manejador(int signal, void *manejador) {
  struct sigaction act;
  struct sigaction old_act;

  int i;

  act.sa_handler = manejador;
  sigemptyset(&act.sa_mask);
  act.sa_flags = SA_RESTART;
  sigaction(signal, &act, &old_act);
}

int main(int argc, char **argv) {

  struct itimerval it;
  int segundos;

/*
Inicializar el manejador con la funcion ini_manejador
*/
/*****************************************************************/
    ini_manejador(SIGALRM, alarma);
/*****************************************************************/

  segundos = atoi(argv[1]);
  it.it_value.tv_sec = segundos;
  it.it_value.tv_usec = 0;
  it.it_interval.tv_sec = segundos;
  it.it_interval.tv_usec = 0;

/*
Inicializar el reloj de pared (wall-clock) mediante setitimer
*/
/*****************************************************************/
   if(setitimer(ITIMER_REAL,&it,NULL) == -1){
    perror("setitimer");
    exit(1);
   }
/*****************************************************************/

  while(contador<5){};
  printf("Recibidas %d veces la señal de alarma\n",contador);
  return 0;
}


/*
badr@badr:~/Escritorio/ASO/Ejercicio3/Ejercicio3.3$ ./alarma.elf 2
Se acabo la cuenta
Se acabo la cuenta
Se acabo la cuenta
Se acabo la cuenta
Se acabo la cuenta
Recibidas 5 veces la señal de alarma
*/
```

In [ ]:

```
%%bash
gcc alarma.c -o alarma.elf
# El funcionamiento de este ejercicio se aprecia mejor en terminal:
# usuarioso@fb6aefc36de3:~/aso-jupyter-public/ejercicios/ejercicio3.2$ ./alarma.elf 2
# Se acabo la cuenta
# Se acabo la cuenta
# Se acabo la cuenta
# Se acabo la cuenta
# Se acabo la cuenta
# Recibidas 5 veces la señal de alarma
```

Ejercicio 5 - minishell2.c Escribe un programa que extienda la shell básica implementada en `minishell.c`, de modo que se capture la señal que se envía a un proceso (padre) cuando alguno de sus hijos finaliza. ¿Qué señal recibe un proceso cuando muere uno de sus procesos hijos?

Respuesta:

Le envia la señal SIGCHLD. ...

In [ ]:

```
%%writefile minishell2.c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <errno.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>

extern int errno;

char ** parse(char *line) {
  static char delim[] = " \t\n";
  int count = 0;
  char * p;
  static char **scr;

  if (*line == '\n')
    return NULL;

  for (p = (char *) strtok(line, delim); p; p = (char *) strtok(NULL, delim)) {
    scr = (char **) realloc(scr, (count + 2) * sizeof(char *));
    scr[count++] = p;
  }
  scr[count] = NULL;

  return scr;
}

void handler(int signo) {
  int errno_bck;
  pid_t pid;
  char m[80]="\nMurió un hijo\n";
  int status;
  int fin=0;

  write(1, m, 80);

  errno_bck = errno;

  while(!fin) {
    pid = waitpid(-1, &status, WNOHANG);
    if (pid == 0)
      fin = 1;
    else if (pid == -1 && errno == ECHILD)
      fin = 1;
    else if (pid == -1) {
```

```
      perror("waitpid");
      abort();
    }
  }

  errno = errno_bck;
}

int main() {
  char **argumentos;
  char comando[80];
  pid_t pid, pid_ret;
  struct sigaction act;
/*
Inicializar act convenientemente
*/
/*****************************************************************/

/*****************************************************************/

  setbuf(stdout, NULL);
  printf("\n");
  while (1) {
    do {
      printf("> ");
      fgets(comando, 80, stdin);
      argumentos = parse(comando);
    } while(argumentos == NULL);

    pid = fork();

    switch(pid) {
      case -1:
        perror("fork");
        exit(1);
        break;
      case 0:
        execvp(argumentos[0], argumentos);
        perror("execlp");
        exit(1);
        break;
      default:
        break;
    }
  }
}



/*
badr@badr:~/Escritorio/ASO/Ejercicio3/Ejercicio3.3$ gcc minishell2.c -o minishell2.elf
badr@badr:~/Escritorio/ASO/Ejercicio3/Ejercicio3.3$ ./minishell2.elf

> ls -l
> total 124
-rw-r--r-- 1 badr badr  1652 may 10 16:40 ' handler.c'
-rw-r--r-- 1 badr badr  1495 may 10 17:42  alarma.c
-rwxr-xr-x 1 badr badr 17168 may 10 17:42  alarma.elf
-rw-r--r-- 1 badr badr  1652 may 10 16:25  handler.c
-rwxr-xr-x 1 badr badr 17048 may 10 16:27  handler.elf
-rw-r--r-- 1 badr badr  1576 may 10 17:29  killer.c
-rwxr-xr-x 1 badr badr 17152 may 10 17:29  killer.elf
-rw-r--r-- 1 badr badr  1787 may 10 17:52  minishell2.c
-rwxr-xr-x 1 badr badr 17648 may 10 17:53  minishell2.elf
-rw-r--r-- 1 badr badr  1595 may 10 16:43  suspender.c
-rwxr-xr-x 1 badr badr 17064 may 10 16:43  suspender.elf

Murió un hijo
^C

*/
```

In [ ]:

```
%%bash
gcc minishell2.c -o minishell2.elf
```

```
# Este programa hay que ejecutarlo en un terminal
# usuarioso@fb6aefc36de3:~/aso-jupyter-public/ejercicios/ejercicio3.2$ ./minishell2.elf
# > ls -l
# > total 132
# -rw-r--r-- 1 usuarioso usuarioso  1470 Apr  6 16:52 alarma.c
# -rwxr-xr-x 1 usuarioso usuarioso  8768 Apr  6 16:52 alarma.elf
# -rw-r--r-- 1 usuarioso usuarioso  1793 Apr  6 17:22 bloqueo.c
# -rwxr-xr-x 1 usuarioso usuarioso 12952 Apr  6 17:29 bloqueo.elf
# -rw-r--r-- 1 usuarioso usuarioso  1631 Apr  6 15:46 handler.c
# -rwxr-xr-x 1 usuarioso usuarioso  8648 Apr  6 15:46 handler.elf
# -rw-r--r-- 1 usuarioso usuarioso  1576 Apr  6 16:36 killer.c
# -rwxr-xr-x 1 usuarioso usuarioso  8752 Apr  6 16:36 killer.elf
# -rw-r--r-- 1 usuarioso usuarioso  1794 Apr  6 17:05 minishell2.c
# -rwxr-xr-x 1 usuarioso usuarioso 13344 Apr  6 17:05 minishell2.elf
# -rw-r--r-- 1 usuarioso usuarioso  1591 Apr  6 15:56 suspender.c
# -rwxr-xr-x 1 usuarioso usuarioso  8672 Apr  6 15:56 suspender.elf
# Murió un hijo
# > ^C
# usuarioso@fb6aefc36de3:~/aso-jupyter-public/ejercicios/ejercicio3.2$
```

Ejercicio 6 - bloqueo.c A partir del esqueleto de programa propuesto en el fichero `bloqueo.c` escribe un programa que se comporte de la siguiente forma:

- El programa recibirá un sólo argumento, que indica los segundos a dormir.
- Bloqueará las señales SIGINT y SIGTSTP, y ejecutará el comando sleep con tantos segundos como ha indicado el usuario.
- Durante esta fase, las señales correspondientes permanecerán bloqueadas.
- Al finalizar la ejecución de sleep, se consultarán las señales pendientes.
- Si se ha recibido una señal SIGINT, el programa mostrará un mensaje por pantalla indicando esta situación.
- Si se ha recibido una señal SIGTSTP, el programa desbloqueará dicha señal y continuará. ¿Qué pasa a continuación? ¿Cómo salir del nuevo bloqueo?

Respuesta:

...

In [ ]:

```
%%writefile bloqueo.c
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char **argv) {
  sigset_t grupo;
  sigset_t pendientes;
  int tiempo = 5; // por defecto 5s

/*
Inicializar el conjunto de señales 'grupo' con las señales de interrupcion SIGINT, y parada SIGTST
P
*/
/*************************************************************/
     sigemptyset(&grupo);
     sigaddset(&grupo,SIGINT);
     sigaddset(&grupo,SIGTSTP);
/*************************************************************/

/*
Bloquear el grupo de señales creado
*/
/*************************************************************/
     sigprocmask(SIG_BLOCK, &grupo, NULL);
/*************************************************************/

  if (argv[1])
    tiempo = atoi(argv[1]);

  printf("Durmiendo %d segundos con (sleep)\n", tiempo);
  sleep(tiempo);
/*
```

```
Comprobar qué señales (grupo pendientes) se enviaron mientras el proceso estaba dormido
*/
/************************************************************/
     sigpending(&pendientes);
/************************************************************/

  if (sigismember(&pendientes, SIGINT)) {
    printf("Presionaste ctrl+c, no la trato\n");
  }

  if (sigismember(&pendientes, SIGTSTP)) {
    printf("Presionaste ctrl+z, la desbloqueo (a dormir)\n");
/*
Desbloquear la señal SIGTSTP
*/
/************************************************************/
     sigdelset(&grupo,SIGINT);
     //sigdelset(&grupo,SIGTSTP);
     sigprocmask(SIG_UNBLOCK, &grupo, NULL);

/************************************************************/
    printf("Desperté otra vez \n");
  }
  return 0;
}


/*

comentar sigdelset(&grupo,SIGINT); y  descomentar  //sigdelset(&grupo,SIGTSTP);
badr@badr:~/Escritorio/ASO/Ejercicio3/Ejercicio3.3$ ./bloqueo.elf 8
Durmiendo 8 segundos con (sleep)
^ZPresionaste ctrl+z, la desbloqueo (a dormir)
Desperté otra vez.


comentar //sigdelset(&grupo,SIGTSTP); y  descomentar  sigdelset(&grupo,SIGINT);
badr@badr:~/Escritorio/ASO/Ejercicio3/Ejercicio3.3$ ./bloqueo.elf 8
Durmiendo 8 segundos con (sleep)
^Z^Z^CPresionaste ctrl+c, no la trato
Presionaste ctrl+z, la desbloqueo (a dormir)

[10]+  Detenido               ./bloqueo.elf 8

pruba de que la señar se reactiva y mata el proceso (^ C);
*/
```

In [ ]:

```
%%bash
gcc bloqueo.c -o bloqueo.elf
# Este programa hay que ejecutarlo en un terminal
# usuarioso@fb6aefc36de3:~/aso-jupyter-public/ejercicios/ejercicio3.2$ ./bloqueo.elf 8
# Durmiendo 8 segundos con (sleep)
# ^C^C^Z^CPresionaste ctrl+c, no la trato
# Presionaste ctrl+z, la desbloqueo (a dormir)
#
# [6]+  Stopped                 ./bloqueo.elf 8
# usuarioso@fb6aefc36de3:~/aso-jupyter-public/ejercicios/ejercicio3.2$
```

Ejericicio 7. Crea un **Makefile** que compile todos los ejemplos de modo que se generen ejecutables con el mismo nombre que el fuente, pero con extensión **.elf**. El **Makefile** deberá incluir una regla **clean** que borre todos los ficheros **.o** y **.elf** generados.

In [ ]:

```
%%writefile Makefile
CC = gcc
CFLAGS = -g -Wall -Werror
#################################################
all:  handler.elf alarma.elf bloqueo.elf killer.elf minishell2.elf suspender.elf
dup.elf open.elf rw.elf
%.elf: %.o
$(CC) $(CFLAGS) $^ -o $@
%.o: %.c
```

```
%.o: %.c
$(CC) -c $< -o $@
.PHONY: clean
clean:
rm -f *.o *.elf
#################################################
```

In [ ]:

```
%%bash
make clean
make
```