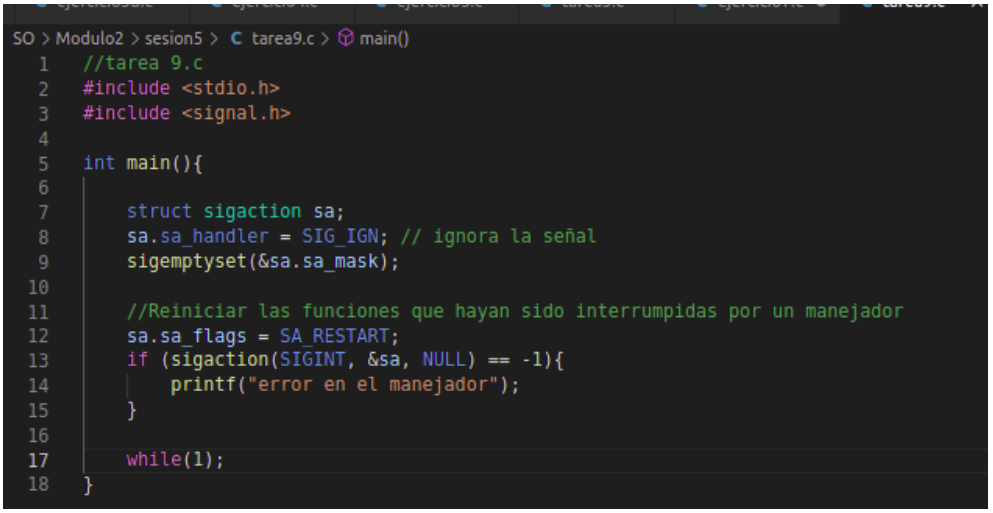
SESIÓN 5 MÓDULO II

tarea 9.c



En este ejercicio, lo que se hace básicamente es, definir un tipo de dato struct sigaction (ya que está sigaction y struct sigaction, y nos interesa el segundo para poder acceder a sa_handler. Con sa_handler, lo que se consigue es especificar qué acción va a llevar a cabo el argumento “signum” de la función sigaction:

int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);

Entonces, con sa (tipo de dato struct sigaction), accedemos a sa_handler y le asignamos SIG_IGN → ignora la señal.

A continuación, para el campo sa_mask del struct, se accede con la función sigemptyset, que lo que hace es inicializar a vacío un conjunto de señales. Y hacemos sa.sa_mask dentro de la función

Con sa accedemos a sa_flags, sa.sa_flags, y le asignamos el valor SA_RESTART, que hace que algunas llamadas del sistema se reinicien debido a que fueron interrumpidas por una señal.

Y ya realizamos la función de sigaction, que si da -1 se producirá un error. Los argumentos que se le pasan son:

- SIGINT→ interrupción del teclado Ctrl + C, de tipo int.
- &sa → el struct sigaction definido anteriormente. Como act no es null, la señal se guarda en act.
- NULL →oldact es null, entonces la señal anterior no se guarda en oldact.

tarea10.c

SO > Modulo2 > sesion5 > C tarea10.c > main()

```c
// tarea10.c
#include <stdio.h>
#include <signal.h>

static int s_recibida=0;

static void handler (int signum){
    printf("\n Nueva acción del manejador \n");
    s_recibida++;
}

int main(){

    struct sigaction sa;
    sa.sa_handler = handler; // establece el manejador a handler
    sigemptyset(&sa.sa_mask);

    //Reiniciar las funciones que hayan sido interrumpidas por un manejador
    sa.sa_flags = SA_RESTART;
    if (sigaction(SIGINT, &sa, NULL) == -1){
        printf("error en el manejador");
    }

    while(s_recibida<3);
}
```

En este ejercicio lo que se hace básicamente es lo siguiente:

- Se crea una función handler, que básicamente será el manejador.
- en el main, se crea otro struct sigaction sa. Accede al campo sa_handler y le asigna la función handler creada anteriormente.
- Se accede al campo sa_mask con sigemptyset, que inicializa a vacío el conjunto de señales.
- Se accede a sa_flags y se le asigna SA_RESTART, para reiniciar las llamadas al sistema que hayan sido interrumpidas.

- Por último a la función sigaction se le pasa SIGINT, interrupción del teclado con Ctrl + C; &sa, que como no es NULL el valor de la señal se guardará ahí; NULL, el campo oldact, ahí no se guarda la señal pues es NULL.

Por tanto, cuando lo ejecutamos nos damos cuenta de que hasta que no se haga ctrl + C durante 3 veces, el programa no termina.
Se llama al sistema sigaction. y se establece un manejador para la señal SIGINT, que se genera cuando se hace ctrl + C.

```
morris@morris-GL63-8RD:~/Documentos/SO/Modulo2/sesion5$ ./tarea10
^C
 Nueva acción del manejador
^C
 Nueva acción del manejador
^C
 Nueva acción del manejador
```

Ejercicio 1. Compila y ejecuta los siguientes programas y trata de entender su funcionamiento.

envioSignal.c

```c
#include <sys/types.h> //POSIX Standard: 2.6 Primitive System Data Types // <sys/typ
#include <limits.h> //Incluye <bits/posix1_lim.h> POSIX Standard: 2.9.2 //Minimum //
#include <unistd.h> //POSIX Standard: 2.10 Symbolic Constants <unistd.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <errno.h>

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

    int pid; //he tenido que cambiar long int por int porque sino me daba error
    int signal;
    if(argc<3) {
        printf("\nSintaxis de ejecución: envioSignal [012] <PID>\n\n");
        exit(-1);
    }

    pid= strtol(argv[2],NULL,10);

    if(pid == LONG_MIN || pid == LONG_MAX){

        if(pid == LONG_MIN)
            printf("\nError por desbordamiento inferior LONG_MIN %d",pid);

        else
            printf("\nError por desbordamiento superior LONG_MAX %d",pid);

        perror("\nError en strtol");
        exit(-1);
    }

    signal=atoi(argv[1]);
    switch(signal) {

        case 0: //SIGTERM
            kill(pid,SIGTERM);
            break;

        case 1: //SIGUSR1
            kill(pid,SIGUSR1);
            break;

        case 2: //SIGUSR2
            kill(pid,SIGUSR2);
            break;

        default : // not in [012]
            printf("\n No puedo enviar ese tipo de señal");
    }
}
```

En este programa, se pretende pasar como argumento 2 cosas:

- Número de señal: puede ser 0, 1, 2. En otro caso se iría al case default. La 0 significa que se terminará el proceso (SIGTERM), la 1 significa que se enviará la señal SIGUSR1 y la 2 SIGUSR2.
- PID del programa que recibe la señal, que se ejecuta antes en background &, y nos muestra su PID que le pasamos. Se encuentra esperando a que llegue la señal.

Obtenemos su PID sacándolo del 2 argumento con strtol, para pasarlo a long int, y se comprueba que no sobrepasa ni LONG_MIN ni LONG_MAX.

Obtenemos su señal con el primer argumento que le pasamos, con atoi. Y ya en función del primer argumento que le pasemos, hará un caso del switch u otro.

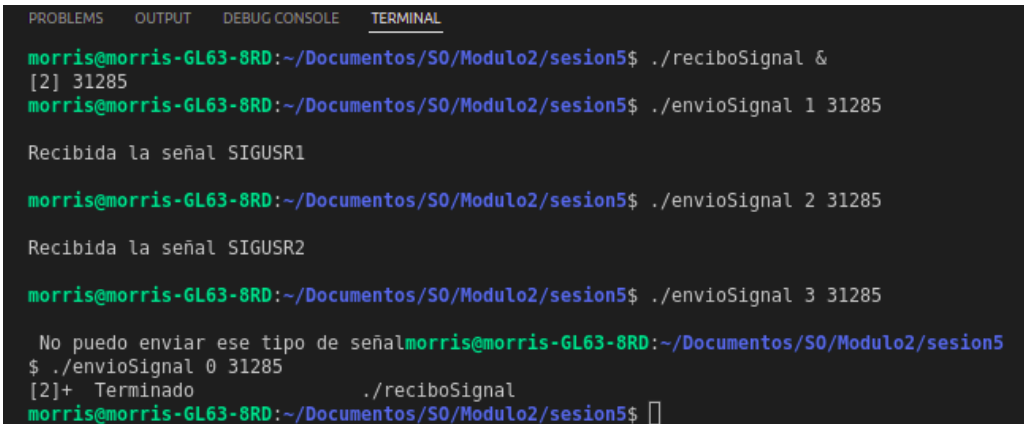
reciboSignal.c

```c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <signal.h>
#include <errno.h>
#include <stdlib.h>

static void sig_USR_hdlr(int sigNum){

    if(sigNum == SIGUSR1)
        printf("\nRecibida la señal SIGUSR1\n\n");

    else if(sigNum == SIGUSR2)
        printf("\nRecibida la señal SIGUSR2\n\n");
}

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

    struct sigaction sig_USR_nact;
    if(setvbuf(stdout, NULL, _IONBF,0)){
        perror("\nError en setvbuf");
    }

    //Inicializar la estructura sig_USR_na para especificar la nueva acción para la
    sig_USR_nact.sa_handler = sig_USR_hdlr;

    //'sigemptyset' inicia el conjunto de señales dado al conjunto vacío.
    sigemptyset (&sig_USR_nact.sa_mask);
    sig_USR_nact.sa_flags = 0;

    //Establecer mi manejador particular de señal para SIGUSR1
    if( sigaction(SIGUSR1,&sig_USR_nact,NULL) <0){
        perror("\nError al intentar establecer el manejador de señal para SIGUSR1");
        exit(-1);
    }

    //Establecer mi manejador particular de señal para SIGUSR2
    if( sigaction(SIGUSR2,&sig_USR_nact,NULL) <0){
        perror("\nError al intentar establecer el manejador de señal para SIGUSR2");
        exit(-1);
    }

    //Que significa??
    for(;;){
    }
}
```

En este programa se implementa un manejador de la señal, sig_USR_hdlr, en el cual si recibe la señal SIGUSR1 o SIGUSR2 imprime eso por pantalla.

Para acceder a todos los campos de sigaction lo que hacemos es declarar un struct sigaction, y con él accedemos a los campos de sa_handler (le asignamos sig_USR_hdlr), a través del cual nos mostrará los mensajes dependiendo de si los números introducidos en el programa anterior sean 1, 2; sa_mask a través de sigemptyset; sa_flags.

Al ejecutar el programa, me sale lo siguiente:



Ejercicio 2. Escribe un programa en C llamado contador, tal que cada vez que reciba una señal que se pueda manejar, muestre por pantalla la señal y el número de veces que se ha recibido ese tipo de señal, y un mensaje inicial indicando las señales que no puede manejar. En el cuadro siguiente se muestra un ejemplo de ejecución del programa.

```c
#include <sys/types.h> //POSIX Standard: 2.6 Primitive System Data Types // <sys/typ
#include <limits.h> //Incluye <bits/posix1_lim.h> POSIX Standard: 2.9.2 //Minimum //
#include <unistd.h> //POSIX Standard: 2.10 Symbolic Constants <unistd.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <errno.h>

static int count_signals[65];
static int signals[65];

static void handler(int sigNum){
    if (sigNum > 64 || sigNum < 0){
        printf("El programa no puede capturar la señal %d\n", sigNum);
        raise(SIGSTOP); //kill(getpid(), SIGSTOP)
    }

    if (signals[sigNum] == 0){//Si es 0, la señal se puede manejar.
        count_signals[sigNum]++;
        printf("La señal %d, se ha escrito %d veces.\n", sigNum, count_signals[sigNu
    }

    else{
        printf("La señal %d es una señal invalida\n", sigNum);
        raise(SIGSTOP);
    }
}

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

    struct sigaction sa;

    if (setvbuf(stdout, NULL, _IONBF, 0)){
        perror("\nError en setvbuf");
    }

    //Inicializo handler
    sa.sa_handler = handler;

    //Inicializamos señales vacías
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;

    //Establecemos los 64 manejadores
    for (int i=0; i<65; i++){
        if (sigaction(i, &sa, NULL) < 0){
            printf("No puedo manejar la señal %d\n", i);
            signals[i] = 1; //si se asigna 1 a esa señal, no se puede manejar.
        }
    }

    printf("Esperando el envío de señales...\n");
    while(1);
}
```

El programa hace lo siguiente:

- Declaro 2 arrays, uno que almacena los números de señales, y otro que almacena las veces que se han enviado ese número de señales

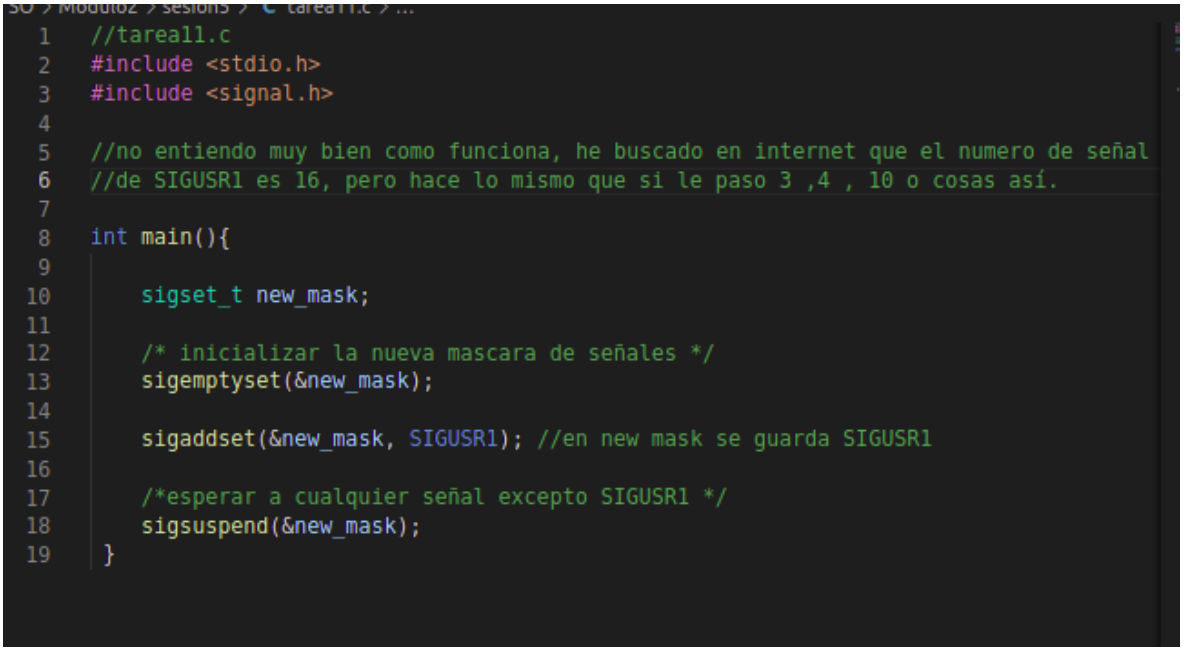
- Primero implemento un manejador de señales, en el cual si el número de señal no se encuentra entre 0 y 64, no se puede capturar esa señal. Si se encuentra en ese rango, ese número de señal lo inicializamos a 0, por lo que sí se puede manejar, y además incrementamos en 1 su contador, ya que se ha enviado una vez.

- Después en el main me declaro un struct sigaction para rellenar todos sus campos correspondientes, y comenzamos a establecer sus manejadores.

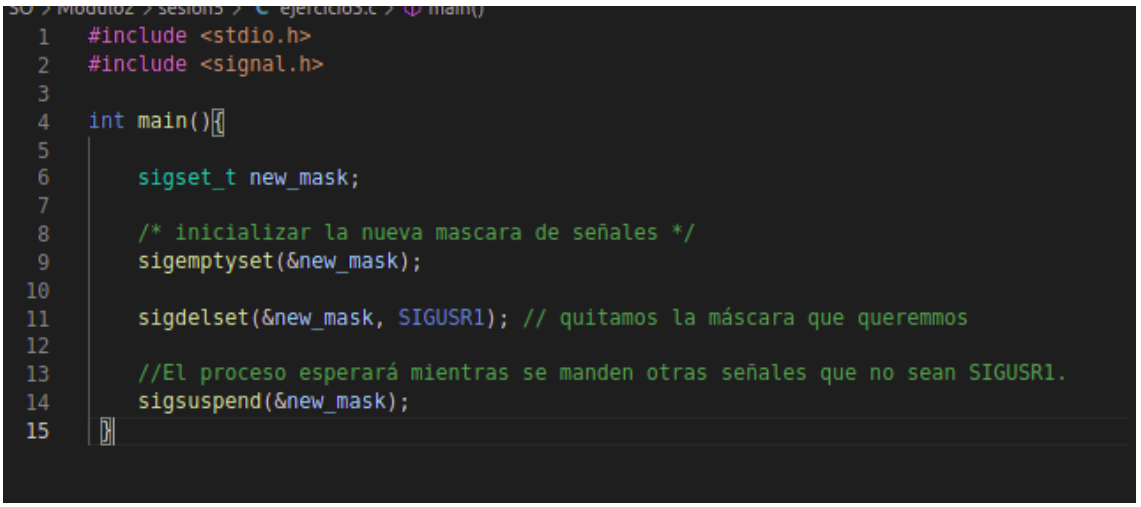
```
morris@morris-GL63-8RD:~/Documentos/SO/Modulo2/sesion5$ ./contador &
[1] 48191
morris@morris-GL63-8RD:~/Documentos/SO/Modulo2/sesion5$ No puedo manejar la señal 0
No puedo manejar la señal 9
No puedo manejar la señal 19
No puedo manejar la señal 32
No puedo manejar la señal 33
Esperando el envío de señales...
kill -2 48191
La señal 2, se ha escrito 1 veces.
morris@morris-GL63-8RD:~/Documentos/SO/Modulo2/sesion5$ kill -2 48191
La señal 2, se ha escrito 2 veces.
morris@morris-GL63-8RD:~/Documentos/SO/Modulo2/sesion5$ kill -3 48191
La señal 3, se ha escrito 1 veces.
morris@morris-GL63-8RD:~/Documentos/SO/Modulo2/sesion5$ kill -19 48191

[1]+  Detenido                ./contador
morris@morris-GL63-8RD:~/Documentos/SO/Modulo2/sesion5$
```

Al ejecutarlo en bg, hacemos kills de los números comprendidos entre 0 y 64, y va calculando cuántas veces se han escrito, pero si hacemos un kill de las señales que no se pueden manejar, directamente se detiene. También podría detenerse haciendo kill -KILL PID.

tarea11.c
En el siguiente ejemplo se suspende la ejecución del proceso actual hasta que reciba una señal distinta de SIGUSR1.



Ejercicio 3. Escribe un programa que suspenda la ejecución del proceso actual hasta que se reciba la señal SIGUSR1. Consulta en el manual en línea sigemptyset para conocer las distintas operaciones que permiten configurar el conjunto de señales de un proceso.



```c
#include <stdio.h>
#include <signal.h>

int main(){

    sigset_t new_mask;

    /* inicializar la nueva mascara de señales */
    sigemptyset(&new_mask);

    sigdelset(&new_mask, SIGUSR1); // quitamos la máscara que queremmos

    //El proceso esperará mientras se manden otras señales que no sean SIGUSR1.
    sigsuspend(&new_mask);
}
```