Departamento de Informática
Grado en Ingeniería Informática
Operating Systems

Signals and Alarms Exercises

Exercise 1.

Implement a C program in which a parent process creates a child process. The parent, 10 seconds after the creation of the child, must send the SIGUSR1 signal to the child process, and then terminate its execution.

The child process, when it receives the signal, should display the following message on the screen: "SIGNAL RECEIVED" and then terminate its execution.

SOLUTION:

SOLUTION: ejemSenal0.c

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

void capturaSenal1 (int s){
        printf ("SIGNAL RECEIVED in child: %d\n",s);
}

main (){
  int i;
  struct sigaction sa1,sa2;

  pid = fork();
  if (pid ¡= 0){
      sleep (10);
      kill (pid, SIGUSR1);
      exit(0);
}
  else { //hijo
    sa1.sa_handler=capturaSenal1;
    sa1.sa_flags=0;
    sigemptyset(&(sa1.sa_mask));
    sigaction (SIGUSR1, &sa1,NULL);

    pause(); // == sleep(999999)
    }
}
```

Exercise 2.

Implement a program that waits 3 seconds for the user to press a key. If the 3 seconds pass without pressing anything, the program ends without continuing to wait.

SOLUTION:

SOLUTION: alarmscanf.c

```
/* este programa espera 3 seg a que el usuario pulse un número y si no lo
hace deja de esperar */

#include <stdio.h>
#include <signal.h>
int finTiempo=0;
void llegoAlarma (int s){
  printf ("Ha llegado la señal de alarma: %d\n",s);
  finTiempo=1;
}
main (){
int i,faltan,n=33;
struct sigaction sa;

      sa.sa_handler= llegoAlarma;
      sa.sa_flags=0;
      sigemptyset (&(sa.sa_mask));
      sigaction (SIGALRM, &sa, NULL);
      alarm(3);
            printf ("Dar un número (tiene 3 segundos) \n" );
            scanf ("%d",&n);
            if (finTiempo)
              printf ("Se acabo el tiempo el número  sera el %d\n",n );
            else
            printf ("número leído %d\n",n );
}
```

Exercise 3.

Implement a program that captures the SIGUSR1 signal and the SIGUSR2 signal. A different treatment must be done for each signal.

SOLUTION:

SOLUTION: ejemSenal1.c

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

void capturaSenal1 (int s){
        printf ("Ha llegado la señal 1: %d\n",s);
        kill (getpid(), SIGUSR2);
}
void capturaSenal2 (int s){
        printf ("Ha llegado la señal 2: %d\n",s);
}
main (){
int i;
struct sigaction sa;

      sa.sa_handler= capturaSenal1;
      sa.sa_flags=0;
      sigemptyset (&(sa.sa_mask));
      sigaction (SIGUSR1, &sa, NULL);
      signal (SIGUSR2, capturaSenal2);
      for  (i=0; i<20; i++){
            sleep(1);
            printf ("%d\n",i);

         }
     kill (getpid(), SIGUSR1);
}
```

Exercise 4.

Implement a program that captures the SIGSEGV signal. In order for it to jump, it must be written in memory position 0.

SOLUTION:

SOLUTION: sigsegv.c

```
#include <stdio.h>
#include <unistd.h>
```

```
#include <signal.h>
#include <stdlib.h>

void capturar_seneal(int segnal){
    printf("Se ha producido un error por intento de ocupacion indebida de
memoria\n");
    signal(SIGSEGV,SIG_DFL);
}

main(void){
    int *p;
    signal(SIGSEGV,capturar_seneal);
    printf ("Ya he colocado el manejador\n");
    p=0;
    printf ("Voy a poner un 5 en la variable\n");
    *p=5;
}
```

Exercise 5.

Implement a program in which the parent process creates a child. The following must be met:

• The parent must capture the SIGUSR1 signal and the SIGCHLD signal. A different treatment must be done for each signal.

• The son falls asleep for 2 seconds, sends the SIGUSR1 signal to the father, and falls asleep 3 seconds before finishing.

SOLUTION:

SOLUTION: ejemSenal2.c

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

void capturaSenal1 (int s){
        printf ("Ha llegado la señal: %d\n",s);
          printf ("tiempo:%d\n", time(NULL));
}
void muerteHijo (int s){
        printf ("Ha terminado el hijo: %d\n",s);
          printf ("tiempo:%d\n", time(NULL));
}
int main (){
int i;
struct sigaction sa1,sa2;
```

```
  if (fork() == 0){
      sleep (10);
      kill ( getppid(), SIGUSR1);
      sleep(3);
  }
  else {
    sa1.sa_handler=capturaSenal1;
    sa1.sa_flags=0;
    sigemptyset(&(sa1.sa_mask));
    sigaction (SIGUSR1, &sa1,NULL);

    sa2.sa_handler=muerteHijo;
    sa2.sa_flags=0;
    sigemptyset(&(sa2.sa_mask));
    sigaction (SIGCHLD, &sa2,NULL);


    pause(); // == sleep(999999)
    pause(); // == sleep(999999)
  }
}
```

Exercise 6.

Show the hierarchy of processes created by this program when invoking it as follows:

./programa 4 5

```
/* Indicar la jerarquía de procesos creados por este programa
 * al invocarlo con programa 4 5 */
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

int seguir =1;

void terminar (int s) {
  seguir =0;
  printf ("%d: alarma\n",getpid());
}

int main (int argc, char *argv[]) {
  int i, numparams, pid, v=33;
  struct sigaction act;

  act.sa_handler = terminar;
  sigemptyset(&act.sa_mask);
  act.sa_flags = 0;
  sigaction(SIGINT, &act, NULL);
  alarm (5);
  i=0;
  numparams=atoi(argv[1]);
  while ( (i< numparams) && seguir){
     if (( pid=fork()) ==0) {
       v++;
       if ( i%2==0 ){
         printf ("Fin en exit Pid %d, v=%d\n",getpid(),v);
         exit (0);
       }
     }
     sleep(2);
     i=i+1;
  }
  printf ("Fin Pid %d, v=%d\n",getpid(),v);
}
```

SOLUTION:

```
padre
        --hijo0
        --hijo1
                ----nieto12
                ----nieto13
        --hijo2
```

Exercise 7.

Implement a program that writes the whole numbers from 1 to 10 on the screen in such a way that the even numbers are written by the father and the odd numbers by the son. The necessary synchronization so that they appear in the correct order is done through signals.

SOLUTION:

```
//Este programa escribe los números enteros del 1 al 10 en pantalla de tal forma
que los números pares los escribe el padre y los impares el hijo.
//La sincronización necesaria para que aparezcan en el orden correcto se hace
mediante señales.
//Se ha utilizado signal en lugar de sigaction
//Hay que tener en cuenta que el envío de señales no es un buen mecanismo de
sincronización y solo se utiliza como ejemplo hasta que se vean métodos mejores

#include <stdio.h>
#include <signal.h>
int n=2;
int m=1;
void capturaSenal1 (int s){ //padre pares
     printf ("Padre %d\n",n);
     n=n+2;
}
void capturaSenal2 (int s){ //hijo impares
     printf ("Hijo %d\n",m);
     m=m+2;
}
main (){
int i,pid;
         signal (SIGUSR1, capturaSenal1);
   signal (SIGUSR2, capturaSenal2);
   pid=fork();
   if (pid !=0)
       sleep(1);
   while ( m<=10 && n<=10) {
       if (pid==0){ //hijo
          pause();
               kill(getppid(),SIGUSR1);
       }
           else { //padre
           kill(pid,SIGUSR2);
               pause();
            }
        }
// return(1);
}
```

Exercise 8.

Write a program:

- to generate a child process, which waits until a SIGINT signal receives a signal, after which it dies,
- the parent process must wait for the previous child process to die and create another child.

The process must run in an infinite loop

SOLUTION:

```
#include <stdio.h>
#include <signal.h>
void tratar_senyal(){
   printf("Hijo muere\n");
   exit(0);
}
main (int argc, char **argv) {
   int pid, status;
   for (;;){
       printf("Padre: creando hijo\n");
       pid = fork();
       if (pid == 0) {
           printf("Hijo: esperando\n");
           signal (SIGINT, tratar_senyal);
           for(;;)
               sleep(1);
       }else{
           signal (SIGINT, SIG_IGN);
           /*
           Es equivalente a:
           struct sigaction act;
           act.sa_handler = SIG_IGN;
           act.flags = 0;
           sigemptyset(&act.sa_mask);
           sigaction(SIGINT, &act, NULL);
           */
           printf("Padre: esperando al hijo %d\n",pid);
           wait(&status);
       }
   }
}
```

Exercise 9

Write a C function on UNIX that allows you to run a command from a program. In case the command cannot be executed, it should give a warning message and return to the program without failure. Possible statement:

int sistema (char *nombre, char *argv)

The execution of the command will be done through a child process. During the execution of this child process, the parent process should not be interrupted by the signals generated from the keyboard. The child process should keep the parent's original signal processing.

SOLUTION

```
#include <signal.h>
int sistema (char *nombre, char *argv)
{
   int pid, status, (*del)(), (*quit)();

   /* padre ignora senyales y */
    /* recupera nombres de rutinas de tratamiento de señales */
   /* para que el mandato se ejecute con el mismo tratamiento de */
   /* señales que había antes de llamar a la función. Es importante */
   /* recordar que las señales que estén capturadas pasaran a tener */
   /* asociadas la acción por defecto después del execvp /*
   del = signal (SIGINT, SIG_IGN);
   quit = signal (SIGQUIT, SIG_IGN);

   pid=fork();
   switch (pid)
   {
       case 0:
           signal (SIGINT, del);
           signal (SIGQUIT, quit);
               .
           /* no se conservan después del execv */
           execvp (nombre, argv);
           perror ("al hacer exec");
               .
           /* si el hijo no puede ejecutar el programa */
           exit (1);
       case -1:
           /* fallo del fork() */
           status = -1;
```

```
            break;
        default:
            /* padre espera el fin del hijo */
            wait(&status);
    }
    signal (SIGINT, del);
    signal (SIGQUIT, quit);
    return(status);
}
```

Exercise 10.

Write a program that prints a message every 10 seconds handling the SIGALRM signal.

Solution

Solution: ejemAlarmRepeat.c

```
#include <stdio.h>

#include <stdlib.h>

#include <signal.h>



void manejador_alarma (int s){

        printf ("Ha llegado la seÒal de alarma: %d\n",s);

}



main (){

        struct sigaction sa;



        sa.sa_handler= manejador_alarma;

        sa.sa_flags=0;

        sigemptyset (&(sa.sa_mask));
        sigaction (SIGALRM, &sa, NULL);


        for(;;){

                alarm(10);

                pause();

        }

        exit(0);

}
```

Exercise 11.

Write a program to install a handler for SIGINT (Ctrl + C) signal that prints a message when the user presses Ctrl + C. Once you have received a number of Ctrl + C equals 10 the process a signal SIGABRT send itself to complete execution.

Solution:

Solution: sigint.c

```
#include <stdio.h>

#include <stdlib.h>

#include <signal.h>



#define         NUM     10

int cont=0;



void manejador_ctrlc (int s){

        printf ("Ha llegado la seÒal de alarma: %d\n",s);
        cont++;

        if(cont==NUM)

                raise(SIGABRT);

}


main (){

        struct sigaction sa;


        sa.sa_handler= manejador_ctrlc;

        sa.sa_flags=0;

        sigemptyset (&(sa.sa_mask));

        sigaction (SIGINT, &sa, NULL);


        for(;;){

                pause();

        }

        exit(0);

}
```

Exercise 12.

Write a program where a process creates 10 threads. Every thread processes simply suspend its execution by calling pause (). Before creating threads the parent establishes a handler for SIGINT signal whose code is simply to print your PID and thread exit. The parent process then sends a SIGINT signal to all the threads to complete their execution.

Solution

Solution: ejempausethreads.c

```
#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

#include <signal.h>


#define         NUM 3


void manejador_ctrlc (int s){

        printf ("Ha llegado la seÒal %d al proceso hijo =%d\n",s,pthread_self());

        pthread_exit(0);

}


int funcion(){

        for(;;){

        }

}

main (){

        int i;

        pid_t pid;

        int status;

        char tecla;

        pthread_t th[NUM];

        struct sigaction sa;
```

```
	sa.sa_handler= manejador_ctrlc;

	sa.sa_flags=0;

	sigemptyset (&(sa.sa_mask));

	sigaction (SIGINT, &sa, NULL);


	for(i=0;i<NUM;i++){

		pthread_create(&th[i],NULL,(void*)funcion,NULL);

	}

	for(i=0;i<NUM;i++){

		printf("Envio seÒal %d a proceso %d\n", SIGINT, th[i]);

		pthread_kill(th[i], SIGINT);

	}

	printf("pulsa tecla\n");

	scanf("%c",&tecla);

	for(i=0;i<NUM;i++){

		pthread_join(th[i],NULL);

	}

	exit(0);

}
```