REPORTE DE LA PRÁCTICA 8

HERNÁNDEZ TAPIA LUIS ENRIQUE

PROFESOR: MONTES CASIANO HERMES FRANCISCO

Escuela Superior de Cómputo, IPN

27 de abril de 2018

Índice general

CAPÍTULO 1

Introducción

Una señal es un "aviso" que puede enviar un proceso a otro proceso. El sistema operativo unix se encarga de que el proceso que recibe la señal la trate inmediatamente. De hecho, termina la línea de código que esté ejecutando y salta a la función de tratamiento de señales adecuada. Cuando termina de ejecutar esa función de tratamiento de señales, continua con la ejecución en la línea de código donde lo había dibujado.

El sistema operativo envía señales a los procesos en determinadas circunstancias. Por ejemplo, si en el programa que se está ejecutando en una shell nosotros apretamos Ctrl-C, se está enviando una señal de terminación al proceso. Este la trata inmediatamente y sale. Si nuestro programa intenta acceder a una memoria no válida (por ejemplo, accediendo al contenido de un puntero a NULL), el sistema operativo detecta esta circunstancia y le envía una señal de terminación inmediata, con lo que el programa "se cae".

CAPÍTULO 2

Desarrollo de la práctica

El programa 11-1 se ejecuta la función *tratar-alarma()* cada 3 segundos. Esta se ha programado para ser un manejador de la señal *SIGALRM*.

Se nos pide escribir el siguiente programa 11-1 y guardarlo como programa11-1.c, para ejecutarl y tratar de matar al procesos con *CTRL-C*.

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


void tratar_alarma(int)
{
        printf("Alarma activada \n");
}

int main(void)
{
        struct sigaction act;
        sigset_t mask;

        /*Especifica el manejador*/
        act.sa_handler = tratar_alarma; // Funcion a ejecutar
        act.sa_flags = 0; // Ninguna accion especifica

        /*Se bloquea la senial 3 SIGQUIT*/
        sigemptyset(&mask);
        sigaddset(&mask, SIGQUIT);
        sigprocmask(SIG_SETMASK, &mask, NULL);
        sigaction(SIGALRM, &act, NULL);
        for (;;)
        {
                alarm(3); /*Se arma el temporizador*/
                pause(); /*Se suspende el proceso hasta que se reciba la senial*/
        }
```

```
30  }
```


Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, D. F. ☎ 57296000 Ext. 52005 ✉ ulises.velez@gmail.com

CAPÍTULO 3

Pruebas

Ejercicio 11.1 Modifique el programa para que se envíe la señal *SIGALARM* cada medio segundo en vez de cada tres segundos (véase ualarm). además modifique la función manejadora de la señal para que imprima el número de señal que le envía el *kernel* a nuestro proceso.

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


void tratar_alarma(int n)
{
        printf("Alarma activada \n");
        printf("%d\n", n);
}

int main(void)
{
        struct sigaction act;
        sigset_t mask;

        /*Especifica el manejador*/
        act.sa_handler = tratar_alarma; // Funcion a ejecutar
        act.sa_flags = 0; // Ninguna accion especifica

        /*Se bloquea la senial 3 SIGQUIT*/
        sigemptyset(&mask);
        sigaddset(&mask, SIGQUIT);
        sigprocmask(SIG_SETMASK, &mask, NULL);
        sigaction(SIGALRM, &act, NULL);
        for (;;)
        {
                //Alarma para que se active, 1000000 microsegundos = 1 seg
                //Entonces 500000 microseg = .5 seg
                ualarm(500000,0); /*Se arma el temporizador*/
```

```
31                      pause(); /*Se suspende el proceso hasta que se reciba la senial*/
32              }
33      }
```

Ejercicio 11.2 Ejecute el programa y presione *(CTRL-)*. Esta combinación envía la señal *SIGQUIT(3)* al proceso, y como podrá observar es una señal ignorada. Modifique el programa para que esta señal provoque su terminación.

```
1   #include <stdio.h>
2   #include <signal.h>
3   #include <unistd.h>
4   #include <stdlib.h>
5
6   void tratar_alarma(int sig)
7   {
8           printf("Alarma activada \n");
9           printf("%d\n", sig);
10  }
11
12  int main(void)
13  {
14          struct sigaction act;
15          sigset_t mask;
16
17          /*Especifica el manejador*/
18          act.sa_handler = tratar_alarma; // Funcion a ejecutar
19          act.sa_flags = 0; // Ninguna accion especifica
20
21          /*Se bloquea la senial 3 SIGQUIT*/
22          sigemptyset(&mask);
23          //sigaddset(&mask, SIGQUIT);
24          sigprocmask(SIG_SETMASK, &mask, NULL);
25          sigaction(SIGALRM, &act, NULL);
26          for (;;)
27          {
28                  //Alarma para que se active, 1000000 microsegundos = 1 seg
29                  //Entonces 500000 microseg = .5 seg
30                  ualarm(500000,0); /*Se arma el temporizador*/
31                  pause(); /*Se suspende el proceso hasta que se reciba la senial*/
32          }
33  }
```

Ejercicio 11.3 Modifique el programa 11-1 para evitar que se detenga con la señal 18 *SIGTSTP* y que además imprima el mensaje "Me han intentado detener".

```
1   #include <stdio.h>
2   #include <signal.h>
3   #include <unistd.h>
4   #include <stdlib.h>
5
6   void tratar_alarma(int sig)
7   {
8           if (sig == 20 || sig == 18 ||sig == 24)
9           {
10                  printf("Me han intentado detener\n");
```


Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, D. F. ☎ 57296000 Ext. 52005 ✉ ulises.velez@gmail.com

```
        }
        printf("Alarma activada \n");
        printf("%d\n", sig);
}

int main(void)
{
        struct sigaction act;
        sigset_t mask;

        /*Especifica el manejador*/
        act.sa_handler = tratar_alarma; // Funcion a ejecutar
        act.sa_flags = 0; // Ninguna accion especifica

        /*Se bloquea la senial 3 SIGQUIT*/
        sigemptyset(&mask);

        sigaddset(&mask, SIGQUIT);//Aniadimos a la lista que evita
        //sigaddset(&mask, SIGTSTP);//Aniadimos a la lista que evita

        sigprocmask(SIG_SETMASK, &mask, NULL);
        sigaction(SIGALRM, &act, NULL);
        sigaction(SIGTSTP,&act,NULL);
        for (;;)
        {
                //Alarma para que se active, 1000000 microsegundos = 1 seg
                //Entonces 500000 microseg = .5 seg
                ualarm(500000,0); /*Se arma el temporizador*/
                pause(); /*Se suspende el proceso hasta que se reciba la senial*/
        }
}
```

Ejercicio 11.4 Modifique el programa para que ninguna señal lo pueda detener (*sigfillset()*)

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

void tratar_alarma(int sig)
{
        printf("Alarma activada \n");
        printf("%d\n", sig);
}

int main(void)
{
        struct sigaction act;
        sigset_t mask;

        /*Especifica el manejador*/
        act.sa_handler = tratar_alarma; // Funcion a ejecutar
        act.sa_flags = 0; // Ninguna accion especifica

        /*Se bloquea la senial 3 SIGQUIT*/
```


Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, D. F. ☎ 57296000 Ext. 52005 ✉ ulises.velez@gmail.com

```
        sigemptyset(&mask);
        //Aniadimos a la lista que evita
        sigaddset(&mask, SIGQUIT);
        //hacemos un fill de todas las seniales para evitarlas
        sigfillset(&mask);

        sigprocmask(SIG_SETMASK, &mask, NULL);
        sigaction(SIGALRM, &act, NULL);
        sigaction(SIGTSTP,&act,NULL);
        for (;;)
        {
                //Alarma para que se active, 1000000 microsegundos = 1 seg
                //Entonces 500000 microseg = .5 seg
                ualarm(500000,0); /*Se arma el temporizador*/
                pause(); /*Se suspende el proceso hasta que se reciba la senial*/
        }
}
```

Ejercicio 11.5 En el siguiente programa el padre crea un hijo que se va a ejecutar un mando recibido en la línea de comandos y espera a que termine. Si el proceso hijo no termina antes de que pasen 5 segundos, el proceso padre debe matar al proceso hijo con la función *kill()*.

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

pid_t pid;

int main(int argc, char const *argv[])
{
        struct sigaction act;
        sigset_t mask;
        /*Especifica el manejador*/
        act.sa_flags = 0; // Ninguna accion especifica

        int pid, status, salida;
        char comando[100] = "";
        /*Se cre el proceso hijo*/
        pid = fork();
        switch(pid)
        {
                case -1:
                        exit(-1);
                case 0:
                /*El proceso hijo ejecuta el comando solicitado*/
                        sprintf(comando,argv[1],getenv("USER"));
                        salida = system(comando);
                        //sleep(6);
                default:
                /*Establece el manejador*/
                        if (wait(&status) >=5 )
                        {
```


Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, D. F. ☎ 57296000 Ext. 52005 ✉ ulises.velez@gmail.com

```
34                                      kill(pid,SIGKILL);
35                              }
36                      /*Espera al proceso hijo*/
37                              wait(&status); /*Vease con man para mas detalles sobre
                                    wait*/
38              }
39              exit(0);
40              return 0;
41      }
```


Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, D. F. ☎ 57296000 Ext. 52005 ✉ ulises.velez@gmail.com

CAPÍTULO 4

Conclusión

La comunicación rápida avisa a un proceso padre que un proceso hijo termina con un éxito, SIGCHLD Abortar un proceso pulsando las teclas ctrl C, SIGINT Para matar a un proceso con el kill, SIGKILL Para avisar a un proceso que ha finalizado una alarma, SIGALRM Para despertar a un proceso que estaba en pausa Cuando un proceso recibe una señal, se interrumpe su ejecución, se almacena su estado para posteriormente reanudar su ejecución, se pasa a ejecutar la función que atiende esa señal, esta función esta definida en el proceso receptor, una vez finalizada esta función se reanuda la ejecución del proceso en el punto que se interrumpió.

1) El estado del proceso se guarda en su stack.
2) Se ejecuta el manejador de la señal.
3) Se recupera el estado del proceso y se continúa.

Las señales son atendidas en modo usuario, si el proceso está en modo núcleo, la señal se añade al conjunto de señales pendientes y se atiende cuando se regresa a modo usuario, esto puede causar un pequeño retraso. Cuando un proceso recibe una señal, si el proceso no se ha preparado para recibirla, el resultado es la muerte del proceso.