SO sesión 4: comunicación de procesos

lunes, 9 de noviembre de 2020 12:11

Para leer en el man	Descripción básica	Opciones
sigaction	Reprograma la acción asociada a un evento concreto	
kill (llamada a sistema)	Envía un evento concreto a un proceso	
sigsuspend	Bloquea el proceso que la ejecuta hasta que recibe un signal (los signals cuyo tratamiento es ser ignorado no desbloquean el proceso)	
sigprocmask	Permite modificar la máscara de signals bloqueados del proceso	
Alarm	Programa el envío de un signal SIGALRM al cabo de N segundos	
Sleep	Función de la librería de C que bloquea al proceso durante el tiempo que se le pasa como parámetro	
/bin/kill (comando)	Envía un evento a un proceso	-L
Ps	Muestra información sobre los procesos del sistema	-o pid,s,cmd,time
Waitpid	Espera la finalización de un proceso	WNOHANG

Kill -SIGKILL pid

```c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
/* ESTE PROCESO PROGRAMA UN TEMPORIZADOR PARA  DENTRO DE 5 SEG
/* LA ACCION POR DEFECTO DEL SIGALRM ES ACABAR EL PROCESO */
int main (int argc,char * argv[]) {
    sigset_t mask;
    sigemptyset(&mask);
    sigaddset(&mask, SIGALRM);
    sigprocmask(SIG_BLOCK, &mask, NULL);
    alarm(10);
    sigfillset(&mask);
    sigdelset(&mask, SIGALRM);
    sigdelset(&mask, SIGINT);
    sigsuspend(&mask);
```

```
GUNDOS Y SE BLOQUEA A ESPERAR QUE LLEGUE */
```

```
    exit(1);
}
```

Ejemplo 1: Programa una alarma de 10 segundos y suspende el programa, cuando recibe la alarma finaliza el programa.

```
int segundos = 0;
/* FUNCION DE ATENCION AL SIGNAL SIGALRM */
void funcion_alarma(int s) {
    char buff[256];
    segundos=segundos+10;
    sprintf(buff, "ALARMA pid=%d: %d segundo
    write(1, buff, strlen(buff));
}
int main (int argc,char * argv[]) {
    struct sigaction sa;
    sigset_t mask;
    sigemptyset(&mask);
    sigaddset(&mask, SIGALRM);
    sigprocmask(SIG_BLOCK,&mask, NULL);
    /* REPROGRAMAMOS EL SIGNAL SIGALRM */
    sa.sa_handler = &funcion_alarma;
    sa.sa_flags = SA_RESTART;
    sigfillset(&sa.sa_mask);
    if (sigaction(SIGALRM, &sa, NULL) < 0) e
    while (segundos < 100) {
        alarm(10);  /* Programamos la alarma
        /* Nos bloqueamos a esperar que nos
        sigfillset(&mask);
        sigdelset(&mask, SIGALRM);
        sigdelset(&mask, SIGINT);
        sigsuspend(&mask);
    }
```

señal de la

```
s\n",getpid(),segundos);
```

```
rror_y_exit("sigaction", 1);

 para dentro de 10 segundos */
llegue un evento */
```

```
    }
    exit(1);
}
```

Programa una alarma cada 10 segundos durante 100 segundos

Signal indica la señal, signal == SIGALRM => quiere

```
int segundos=0;
void funcion_alarma(int signal) {
    char buff[256];
    segundos=segundos+10;
    sprintf(buff,"ALARMA pid=%d: %d segundos
    write(1, buff, strlen(buff) );
}
int main (int argc,char * argv[]) {
    struct sigaction sa;
    sigset_t mask;
    /* EVITAMOS QUE NOS LLEGUE EL SIGALRM FUE
    sigemptyset(&mask);
    sigaddset(&mask, SIGALRM);
    sigprocmask(SIG_BLOCK,&mask, NULL);
    /* REPROGRAMAMOS EL SIGNAL SIGALRM */
    sa.sa_handler = &funcion_alarma;
    sa.sa_flags = SA_RESTART;
    sigfillset(&sa.sa_mask);
    if (sigaction(SIGALRM, &sa, NULL) < 0) er
    if (fork() < 0) error_y_exit("fork", 1);
    while (segundos < 100) {
        alarm(10);
        sigfillset(&mask);
```

decir que es una señal de alarma

```
\n",getpid(),segundos);


ERA DEL SIGSUSPEND */


rror_y_exit("sigaction", 1);
```

```
        sigdelset(&mask, SIGALRM);
        sigdelset(&mask, SIGINT);
        sigsuspend(&mask);
    }
    exit(1);
}
```

La tabla de reprogramación de signals se hereda

```
int main (int argc,char * argv[]) {
    struct sigaction sa;
    sigset_t mask;
    /* EVITAMOS TRATAR EL SIGALRM FUERA DEL
    sigemptyset(&mask);
    sigaddset(&mask, SIGALRM);
    sigprocmask(SIG_BLOCK,&mask, NULL);
    /* REPROGRAMAMOS EL SIGNAL SIGALRM */
    sa.sa_handler = &funcion_alarma;
    sa.sa_flags = SA_RESTART;
    sigfillset(&sa.sa_mask);
    if (sigaction(SIGALRM, &sa, NULL) < 0) e
    switch (fork()) {
        case -1: error_y_exit("fork", 1);
        case 0: alarm(10);
    }
    while (segundos < 100) {
        sigfillset(&mask);
        sigdelset(&mask, SIGALRM);
        sigdelset(&mask, SIGINT);
        sigsuspend(&mask);
        alarm(10);
```

```
SIGSUSPEND */


rror_y_exit("sigaction", 1);
```

```
    }
    exit(1);
}
```

Los SISGALRM son recibidos únicamente por el proceso que los ha generado. Es decir, si
un alarm de 10 segundos, su hijo no recibirá la alarma. Y viceversamente.
Alarm(0) retorna segundos restantes de la alarma

Al crear un proceso el hijo HEREDA la tabla
Al mutar, su tabla de signals se pone por de
los signals por tratar se mantienen.
WAITPID => forma bloqueante

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

int contador = 0;
int hijos = 0;

void error_y_exit(char* msg, int exit_status)
{
   perror(msg);
   exit(exit_status);
}
```

el padre hace

de signals de su padre.
efecto (NO SE HEREDA). Pero

```
void trata_hijo(int s) {
	int pid, exit_code;
	char buff[256];
	while ((pid = waitpid(-1, &exit_code, WNOHANG)) > 0) {
		if (WIFEXITED(exit_code)) {
			int statcode = WEXITSTATUS(exit_code);
			sprintf(buff,"Termina el proceso %d com exit code %d\n",pid,
		}
		else {
			int signcode = WTERMSIG(exit_code);
			sprintf(buff,"Han matado al proceso %d antes de acabar alarr
		}
		write(1, buff, strlen(buff));
		hijos--;
		++contador;
	}
}


void trata_alarma(int s)
{
}

int main(int argc, char* argv[])
{
   int pid, res;
   char buff[256];
   struct sigaction sa;
   sigset_t mask;

   /* Evitamos recibir el SIGALRM fuera del sigsuspend */

   sigemptyset(&mask);
   sigaddset(&mask, SIGALRM);
   sigprocmask(SIG_BLOCK, &mask, NULL);

   for (hijos = 0; hijos < 10; hijos++) {
      sprintf(buff, "Creando el hijo numero %d\n", hijos);
```

, statcode);

m por el signal %d\n",pid, signcode);

```
        write(1, buff, strlen(buff));

        pid = fork();
        if (pid == 0) /* Esta linea la ejecutan tanto el padre como el hijo */
        {

            sa.sa_handler = &trata_alarma;
            sa.sa_flags = SA_RESTART;
            sigfillset(&sa.sa_mask);
            if (sigaction(SIGALRM, &sa, NULL) < 0)
               error_y_exit("sigaction", 1);

            /* Escribe aqui el codigo del proceso hijo */
            sprintf(buff, "Hola, soy %d\n", getpid());
            write(1, buff, strlen(buff));

            alarm(2);
            sigfillset(&mask);
            sigdelset(&mask, SIGALRM);
            sigdelset(&mask, SIGINT);
            sigsuspend(&mask);

            /* Termina su ejecución */
            exit(0);
        } else if (pid < 0) {
            /* Se ha producido un error */
            error_y_exit("Error en el fork", 1);
        }
    }
    /* Esperamos que acaben los hijos */
         sa.sa_handler = &trata_hijo;
    sa.sa_flags = SA_RESTART;
    sigaction(SIGCHLD, &sa, NULL);
    while (hijos > 0);
    sprintf(buff, "Valor del contador %d\n", contador);
    write(1, buff, strlen(buff));
    return 0;
}
```

Tratar la causa de las muertes de los hijos, reprogramando l

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

int contador = 0;
int hijos = 0;

void error_y_exit(char* msg, int exit_status)
{
   perror(msg);
   exit(exit_status);
}

void trata_hijo(int s) {
        int pid, exit_code;
        char buff[256];
        while ((pid = waitpid(-1, &exit_code, WNOHANG)) > 0) {
                if (WIFEXITED(exit_code)) {
                        int statcode = WEXITSTATUS(exit_code);
                        sprintf(buff,"Termina el proceso %d com exit code %d\n",pid, statco
                }
                else {
                        int signcode = WTERMSIG(exit_code);
                        sprintf(buff,"Han matado al proceso %d antes de acabar alarm por e
                }
                write(1, buff, strlen(buff));
                hijos--;
                ++contador;
        }
```

los SIGCHLD

ode);

el signal %d\n",pid, signcode);

```
}


void trata_alarma(int s)
{
}

int main(int argc, char* argv[])
{
   int pid, res;
   char buff[256];
   struct sigaction sa;
   sigset_t mask;
        int pid_vec[10];

   /* Evitamos recibir el SIGALRM fuera del sigsuspend */

   sigemptyset(&mask);
   sigaddset(&mask, SIGALRM);
   sigprocmask(SIG_BLOCK, &mask, NULL);

   for (hijos = 0; hijos < 10; hijos++) {
      sprintf(buff, "Creando el hijo numero %d\n", hijos);
      write(1, buff, strlen(buff));

      pid = fork();
      if (pid == 0) /* Esta linea la ejecutan tanto el padre como el hijo */
      {
         sa.sa_handler = &trata_alarma;
         sa.sa_flags = SA_RESTART;
         sigfillset(&sa.sa_mask);
         if (sigaction(SIGALRM, &sa, NULL) < 0)
            error_y_exit("sigaction", 1);

         /* Escribe aqui el codigo del proceso hijo */
         sprintf(buff, "Hola, soy %d\n", getpid());
         write(1, buff, strlen(buff));

         alarm(2);
         sigfillset(&mask);
         sigdelset(&mask, SIGALRM);
```

```
            sigdelset(&mask, SIGINT);
            sigsuspend(&mask);

            /* Termina su ejecución */
            exit(0);
        } else if (pid < 0) {
            /* Se ha producido un error */
            error_y_exit("Error en el fork", 1);
        } else pid_vec[hijos] = pid;
    }
    /* Esperamos que acaben los hijos */
          sa.sa_handler = &trata_hijo;
    sa.sa_flags = SA_RESTART;
    sigaction(SIGCHLD, &sa, NULL);
          for (int i = 0; i < 10; ++i) kill(pid_vec[i], SIGUSR1);
    while (hijos > 0);
    sprintf(buff, "Valor del contador %d\n", contador);
    write(1, buff, strlen(buff));
    return 0;
}
```

PROTECCIÓN ENTRE PROCESOS:

No se puede enviar signals a procesos de otros usuarios
Muestra error: "bash: kill: (5432) - Operation not permitted"

GESTIÓN DE SIGNALS

Se puede reprogramar todos los signals, menos SIGKILL y SIG
PARA REPROGRAMAR UN SIGNAL POR EL VALOR POR DEFECT
Sa_handler = SIG_DFL => comportamiento por defecto
Sa_flags = SA_RESETHAND => comportamiento por defecto
Sigfillst(&sa_mask)

TRATAMIENTO DE SIGNALS ANIDADO

STOP
TO

TRATAMIENTO DE SIGNALS ANIDADO

Cuando configuramos la reprogramación de un signal elegim
queremos actuar ante la recepción de otro signal durante la e
de nuestro código de tratamiento de signal reprogramado.
Por un lado podemos secuencializar el tratamiento, por lo qu
trata hasta que acabe nuestro código. Simplemente hay que
la máscara "sa_mask", todos aquellos signals que queremos l
durante el tratamiento (poner a 1, secuencializar).
Por otro lado, podemos anidar el tratamiento, por lo que con
tratarse de inmediato el tratamiento del otro signal. En este c
comportamiento lo tendrán aquellos signals que no están en
"sa_mask" (a 0).

RIESGOS DE LA CONCURRENCIA

Cuando se programan aplicaciones con varios procesos concu
se puede asumir nada sobre el orden de ejecución de las inst
de los diferentes procesos. Esto también aplica al envío y rec
signals: no podemos asumir que un proceso recibirá un signa
momento determinado.

os cómo
ejecución

e no se
incluir en
bloquear

mienza a
caso, este
la

urrentes no
rucciones
epción de
l en un