دانشگاه صنعتی اصفهان

دانشکده برق و کامپیوتر

آزمایشگاه سیستم عامل

دستور کار جلسه هفتم

برنامه‌نویسی سیگنال

علی فانیان

زینب زالی

تابستان ۱۳۹۸

مروری کلی بر رفتار signal

- Signal ارتباط بین فرآیندها از راه کنترل رخدادهای خاص و تعریف رفتار فرآیند در قبال رخدادهای تعریف شده است.
- برخلاف socket و pipe نمی توان مقداری را از طریق signal منتقل کرد، این روش تنها برای آگاهی از اتفاق افتادن یک رخداد خاص به کار می رود.
- Signal ها نسبت به pipe و socket پیچیدگی‌های بیشتری دارند و به همین دلیل بایستی با احتیاط به کار برده شوند.
- در لینوکس، ۳۲ سیگنال تعریف‌شده وجود دارد. لیست این سیگنال‌ها را با اجرای دستور kill -l یا man 7 signal می‌توانید مشاهده کنید. جدول زیر تعدادی از این سیگنال‌ها را به ترتیب شماره شناسه نشان می‌دهد.

signal	ID	description
SIGHUP	1	Hangup
SIGINT	2	Interrupt (usually DEL or CTRL-C)
SIGQUIT	3	Quit (usually CTRL-\)
SIGILL	4	Illegal instruction
SIGTRAP	5	Trace trap
SIGABRT	6	Abort program
SIGBUS	7	Bus error
SIGFPE	8	Floating point exception
SIGKILL	9	Kill
SIGUSR1	10	User defined signal #1
SIGSEGV	11	Segmentation fault
SIGUSR2	12	User defined signal #2
SIGPIPE	13	Write to a pipe with no reader
SIGALRM	14	Alarm clock
SIGTERM	15	Terminate (default for kill(1))

دریافت و مدیریت سیگنال

سیگنال دریافت شده به یکی از ۳ راه زیر ارزیابی می شود:

۱. Ignoring

در این حالت، سیگنال دریافت شده ولی هیچ تابعی برای مدیریت آن فراخوانی نمی شود.

۲. Handler Function

سیگنال دریافت شده و یک تابع متناظر برای مدیریت آن فراخوانی می‌شود. در حین اجرای تابع یاد شده ممکن است سیگنال‌های دیگری نیز دریافت شوند که بسته به شرایط می‌توانند آنها نیز تابع متناظر خود را فراخوانی کنند و یا موقتا block شوند.

۳. Default Action

در اینجا رفتار پیش فرض تعیین شده برای همه سیگنال‌ها در همه‌ی فرآیندها اجرا می‌شود.

فراخوانی‌های سیستمی مدیریت Signal

Header

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

sigaction

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

این system call در هر پروسس که فراخوانی شود، رفتار پروسس در هنگام دریافت یک سیگنال مشخص را تغییر می‌دهد. آرگومان اول، سیگنال موردنظر را مشخص می‌کند. آرگومان دوم یک structure است که رفتار موردنظر را تعریف می‌کند و در ادامه توضیح داده می‌شود. آرگومان سوم هم رفتار قبلی را ذخیره می‌کند.

struct sigaction

```
struct sigaction {
   void     (*sa_handler)(int);
   void     (*sa_sigaction)(int, siginfo_t *, void *);
   sigset_t   sa_mask;
   int        sa_flags;
   void     (*sa_restorer)(void);
};
```

این ساختار، یک رفتار را تعریف می‌کند که می‌توان آن را برای اجرا حین رخداد یک سیگنال، با تابع sigaction مشخص کرد. آرگومان اول، اشاره‌گری به یک تابع signal handler است که تابع اصلیی می‌باشد که حین رخداد سیگنال انتظار داریم اجرا شود. آرگومان sa_mask مشخص می‌کند که چه سیگنال‌هایی حین اجرای این رفتار، بلاک شوند. آرگومان sa_flag نیز مجموعه‌ای خاص از flagها را مشخص می‌کند که رفتار سیگنال را تغییر می‌دهند. مجموعه‌ای از این flagها می‌توانند با هم OR شوند.

Handler function

```
void Handler (int signo)
{
switch (signo)
            {
            case SIGINT:
            required action;
            break;
            }
}
```

در این قسمت، قالب کلی یک تابع signal handler که به ساختار بالا داده می‌شود، نشان داده شده است. Signo مشخص می‌کند که سیگنالی که اتفاق افتاده و باعث اجرای این تابع شده است، کدام سیگنال است. از این شناسه سیگنال در پیاده‌سازی handler همان‌طور که نشان داده شده‌است استفاده می‌شود. بدین ترتیب می‌توان برای چند سیگنال متفاوت، از یک تابع signal handler استفاده کرد که در بدنه آن با توجه به شناسه سیگنال رخداده، عملیات متفاوت اجرا شود.

sigset*

```
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);
int sigismember(const sigset_t *set, int signum);
```

با استفاده از این توابع، مجموعه سیگنال‌ها جهت mask کردن (که در sigprocmask استفاده می‌شود، تعریف و ساخته می‌شود (به برنامه نمونه و man مراجعه کنید)

sigprocmask

```
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
```

با استفاده از این تابع، می‌توان سیگنال‌هایی را که برای thread جاری mask شده‌اند (تا فعلاً بلاک شوند و دریافت نشوند)، مشخص کرد یا بدست آورد. با استفاده از آرگومان how می‌توان مشخص کرد که مجموعه جدید mask به قبلی‌ها اضافه شود؟ یا از قبلی‌ها حذف شود یا مجموعه جدید به جای مجموعه قبلی، در نظر گرفته شود. (به man مراجعه کنید)

kill

```
#include <sys/types.h>
#include <signal.h>

int kill(pid_t pid, int sig);
```

با استفاده از تابع kill می‌توان به یک پروسس، سیگنال مشخصی را با استفاده از شناسه آن سیگنال ارسال کرد.

در ادامه مثال‌هایی از نحوه استفاده از سیگنال آمده است. لطفاً مثال‌ها را به دقت اجرا کنید و عمل‌کرد آن‌ها را مشاهده کنید.

مثال ها

Handling Interrupt Signal

```
/*
this program initializes a signal action
assigns a handler to this action and waits for SIGINT (ctrl+c) to be handled
*/

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

// "handler1" is handler function for action1
void handler1(int signo)
{
        switch(signo)
        {
                case SIGINT:
                printf("Interrupt Signal received \n");
                break;
        }
}


int main()
{
        //initializing sigaction structure
        struct sigaction action1;
        action1.sa_handler = handler1;
        action1.sa_flags = 0;
        sigaction(SIGINT,(struct sigaction *) &action1,NULL);
        //runnign forever, while process is sensitive to SIGINT
        while(1);
        return 0;
}
```

Blocking a Signal in Handler

```
/*
this program initializes 2 signal actions, assigns same handler to both actions.
while action1 is handled, SIGUSR2 will be blocked on delivery.
*/

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

#define MAXCHILD 1

// "handler1" is handler function for action1 and action2, returns void
void handler1(int signo)
{
	switch(signo)
	{
	/* handling SIGUSR1 takes one second
	during this time if SIGUSR2 will be blocked on delivery. */
	case SIGUSR1:
		sleep(1);
		printf("SIGUSR1 received \n");
		break;
	case SIGUSR2:
		printf("SIGUSR2 received \n");
		break;
	}

}

int main()
{
	//initializing sigaction structures, action1 and action2 both use handler1
	struct sigaction action1;
	struct sigaction action2;

	sigset_t set1;			//define signal set named "set1"
	sigemptyset(&set1);		//making set1 empty
	sigaddset(&set1, SIGUSR2);	//adding SIGUSR2 to set1

	action1.sa_handler = handler1;
	action1.sa_mask = set1;

	//set1 includes SIGUSR2, it means if during handling action1
	//SIGUSR2 will be blocked on delivery.
	action1.sa_flags = 0;

	action2.sa_handler = handler1;
//	action2.sa_mask = 0;		//no signal has been blocked for action2
	action2.sa_flags = 0;
	int inchild=0;

	//initializng parent process before fork()
	sigaction(SIGUSR1,(struct sigaction *) &action1,NULL);
	sigaction(SIGUSR2,(struct sigaction *) &action2,NULL);

	pid_t parent=getpid();
```

```
	pid_t pid[MAXCHILD];

	for ( int i=0;i<MAXCHILD;i++)
	{
		pid[i]=fork();
		if( pid[i]==0)
		{
			inchild=1;
			break;
		}
	}

	while(inchild==0);

	while(inchild==1)
	{

	//child sends SIGUSR1 and immediately SIGUSR2 to parent every one second
		kill(parent,SIGUSR1);
		kill(parent,SIGUSR2);
		sleep(1);
	}

	return 0;
}
```

Blocking a Signal in Whole Process

```
/*
this program initializes a signal action
assigns a handler to this action and waits.
during whole process SIGUSR2 will be blocked on delivery.
*/

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>
#define MAXCHILD 1

// "handler1" is handler function for action1 and action2, returns void
void handler1(int signo)
{
	switch(signo)
	{
	// handling SIGUSR1 takes one second
	//during this time if SIGUSR2 will be blocked on delivery.
	case SIGUSR1:
		sleep(1);
		printf("SIGUSR1 received \n");
		break;
	case SIGUSR2:
		printf("SIGUSR2 received \n");
		break;
	}

}
int main()
{
	//initializing sigaction structures, action1 and action2 both use handler1
	struct sigaction action1;
	struct sigaction action2;

	sigset_t set1;			//define signal set named "set1"
	sigemptyset(&set1);		//making set1 empty
	sigaddset(&set1, SIGUSR2); //adding SIGUSR2 to set1

	//set1 includes SIGUSR2, it means if SIGUSR2 will be blocked on delivery.
	sigprocmask(SIG_SETMASK, &set1, NULL);

	action1.sa_handler = handler1;
	action2.sa_handler = handler1;

	int inchild=0;

	//initializng parent process before fork()
	sigaction(SIGUSR1,(struct sigaction *) &action1,NULL);
	sigaction(SIGUSR2,(struct sigaction *) &action2,NULL);

	pid_t parent=getpid();
	pid_t pid[MAXCHILD];
```

```
        for ( int i=0;i<MAXCHILD;i++)
                {
                pid[i]=fork();
                if( pid[i]==0)
                        {
                                inchild=1;
                                break;
                        }
                }

        while(inchild==0);

        while(inchild==1)
        {
        //child sends SIGUSR1 and immediately SIGUSR2 to parent every one second

                kill(parent,SIGUSR1);
                kill(parent,SIGUSR2);
                sleep(1);
        }
        return 0;
}
```