دستور کار جلسه ششم

فراخوانی های سیستمی برای ایجاد و مدیریت signal

در این دستورکار توضیحات بیشتری درباره ارتباط بین فرآیندی با استفاده از signal ارائه شده است.

تمامی توابع و فراخوانی های مطرح شده در این دستور کار از دو آدرس زیر آورده شده اند، برای مطالعه جزییات به آنها به صفحات راهنما و یا آدرس های آورده شده مراجعه کنید :

http://www.minix3.org/manpages

http://linux.die.net/man

مروری کلی بر رفتار signal

Signal

وقفه هایی که برای آگاه سازی یک فرآیند به آن ارسال میشود تا آن فرآیند را از روی دادن اتفاقی آگاه کند

- برای ارسال سیگنال به فرآیند از تابع killاستفاده میشود ...به اینصورت که پارامتر اول تابع، فرآیند مورد نظر و پارامتر دوم سیگنال مورد نظر میباشد.
- فرآیند پس از دریافت سیگنال باید عکس العملی از خود نشان دهد .عملی که یک فرآیند در ازای دریافت سیگنال باید انجام دهد توسط تابع signal به آن ارسال میشود...به اینصورت که پارامتر دوم، سیگنال مورد نظر و پارامتر دوم، تابعی است که در ازای دریافت سیگنال فراخوانی میشود...این تابع در سیگنالها تحت عنوان handler شناخته میشود..درواقع handler function عکس العملی است که برنامه در قبال سیگنال از خودش نشان میدهد...
- اگر بخواهیم فرآیند در ازای دریافت سیگنال همان رفتار پیشفرض را داشته باشد و کار خاصی انجام ندهد پارامتر دوم تابع signal را SIG_DFL قرار میدهیم که درواقع عملیات پیشفرضی است که برنامه در قبال سیگنال نشان میدهد که توسط یکی از مقادیری که در جدول زیر آمده است مقدار میگیرد...

signal	num	notes	description
SIGHUP	1	km	Hangup
SIGINT	2	k	Interrupt (usually DEL or CTRL-C)
SIGQUIT	3	kcm	Quit (usually CTRL-\)
SIGILL	4	Kc	Illegal instruction
SIGTRAP	5	Kc	Trace trap
SIGABRT	6	kcm	Abort program
SIGBUS	7	Kc	Bus error
SIGFPE	8	Kc	Floating point exception
SIGKILL	9	k	Kill
SIGUSR1	10	k	User defined signal #1
SIGSEGV	11	Kc	Segmentation fault
SIGUSR2	12	k	User defined signal #2
SIGPIPE	13	k	Write to a pipe with no reader
SIGALRM	14	k	Alarm clock
SIGTERM	15	km	Terminate (default for kill(1))
SIGEMT	16	xKc	Emulator trap
SIGCHLD	17	pi	Child process terminated
SIGCONT	18	pi	Continue if stopped
SIGSTOP	19	ps	Stop signal
SIGTSTP	20	ps	Interactive stop signal
SIGWINCH	21	xi	Window size change
SIGTTIN	22	ps	Background read
SIGTTOU	23	ps	Background write
SIGVTALRM	24	k	Virtual alarm clock
SIGPROF	25	k	Profiler alarm clock

- در Minix این فهرست از `kill -l` و یا `man 2 sigaction` قابل دسترسی ست.

دریافت و مدیریت سیگنال

فراخوانی های سیستمی مدیریت Signal

Header

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

sigaction

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

در استراکچر زیر مقدار sa_mask تعیین کننده سیگنالهایی است که میخواهیم بلاک شوند...

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);
};
```

Handler function

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

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);
```

این تابع جهت بلاک کردن یک سیگنال در کل فرآیند مورد استفاده قرار میگیرد

sigprocmask

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

kill

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

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

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, returns void
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_mask = 0;
        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;

        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;
        action1.sa_mask = 0;
        action1.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;
}
```