linux系統編程之信號（三）：信號安裝、signal、kill，arise講解

一，信號安裝

如果進程要處理某一信號，那麼就要在進程中安裝該信號。安裝信號主要用來確定信號值及進程針對該信號值的動作之間的映射關係，即進程將要處理哪個信號；該信號被傳遞給進程時，將執行何種操作。

linux主要有兩個函數實現信號的安裝：signal()、sigaction()。其中signal()只有兩個參數，不支持信號傳遞信息，主要是用於前32種非實時信號的安裝；而sigaction()是較新的函數（由兩個系統調用實現：sys_signal以及sys_rt_sigaction），有三個參數，支持信號傳遞信息，主要用來與sigqueue() 系統調用配合使用，當然，sigaction()同樣支持非實時信號的安裝。sigaction()優於signal()主要體現在支持信號帶有參數。

二，signal()用法

```
#include <signal.h>

typedef void (*__sighandler_t) (int);

#define SIG_ERR ((__sighandler_t) -1)

#define SIG_DFL ((__sighandler_t) 0)

#define SIG_IGN ((__sighandler_t) 1)

void (*signal(int signum, void (*handler))(int)))(int);
```

如果該函數原型不容易理解的話，可以參考下面的分解方式來理解：

```
typedef void (*sighandler_t)(int)；

sighandler_t signal(int signum, sighandler_t handler));
```

第一個參數指定信號的值，第二個參數指定針對前面信號值的處理，可以忽略該信號（參數設為SIG_IGN）；可以採用系統默認方式處理信號(參數設為SIG_DFL)；也可以自己實現處理方式(參數指定一個函數地址)。

如果signal()調用成功，返回最後一次也就是上一次為安裝信號signum而調用signal()時的handler值；失敗則返回SIG_ERR。

傳遞給信號處理例程的整數參數是信號值，這樣可以使得一個信號處理例程處理多個信號。

man幫助説明：

```
DESCRIPTION
       The behavior of signal() varies across Unix versions, and has also var-
       ied historically across different versions of Linux. Avoid its use:
       use sigaction(2) instead. See Portability below.

       signal() sets the disposition of the signal signum to handler, which is
       either SIG_IGN, SIG_DFL, or the address of a programmer-defined func-
       tion (a "signal handler").

       If the signal signum is delivered to the process, then one of the fol-
       lowing happens:

       * If the disposition is set to SIG_IGN, then the signal is ignored.

       * If the disposition is set to SIG_DFL, then the default action asso-
         ciated with the signal (see signal(7)) occurs.

       * If the disposition is set to a function, then first either the dis-
         position is reset to SIG_DFL, or the signal is blocked (see Porta-
```

bility below), and then handler is called with argument signum. If
invocation of the handler caused the signal to be blocked, then the

signal is unblocked upon return from the handler.

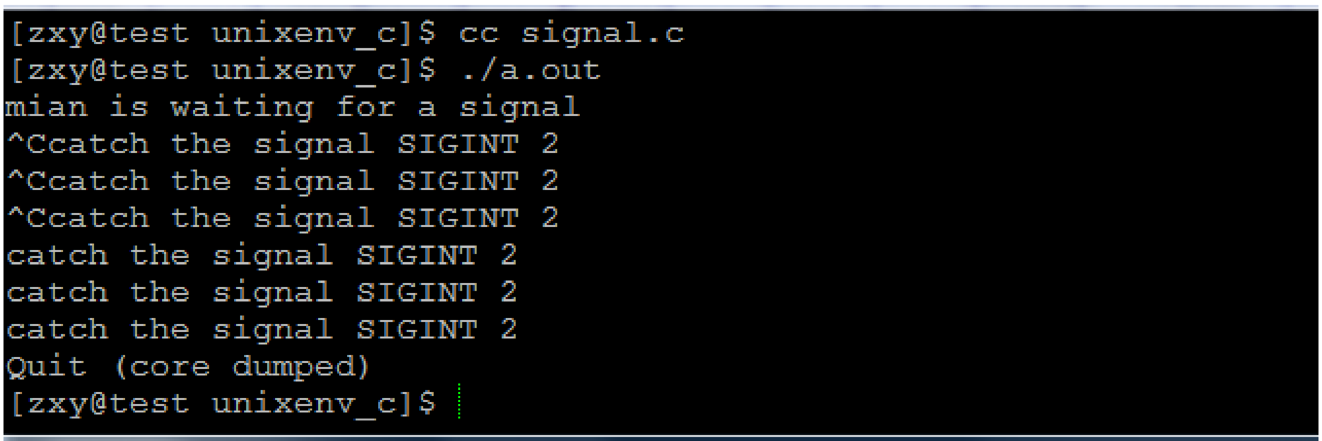
The signals SIGKILL and SIGSTOP cannot be caught or ignored.

RETURN VALUE
signal() returns the previous value of the signal handler, or SIG_ERR
on error.

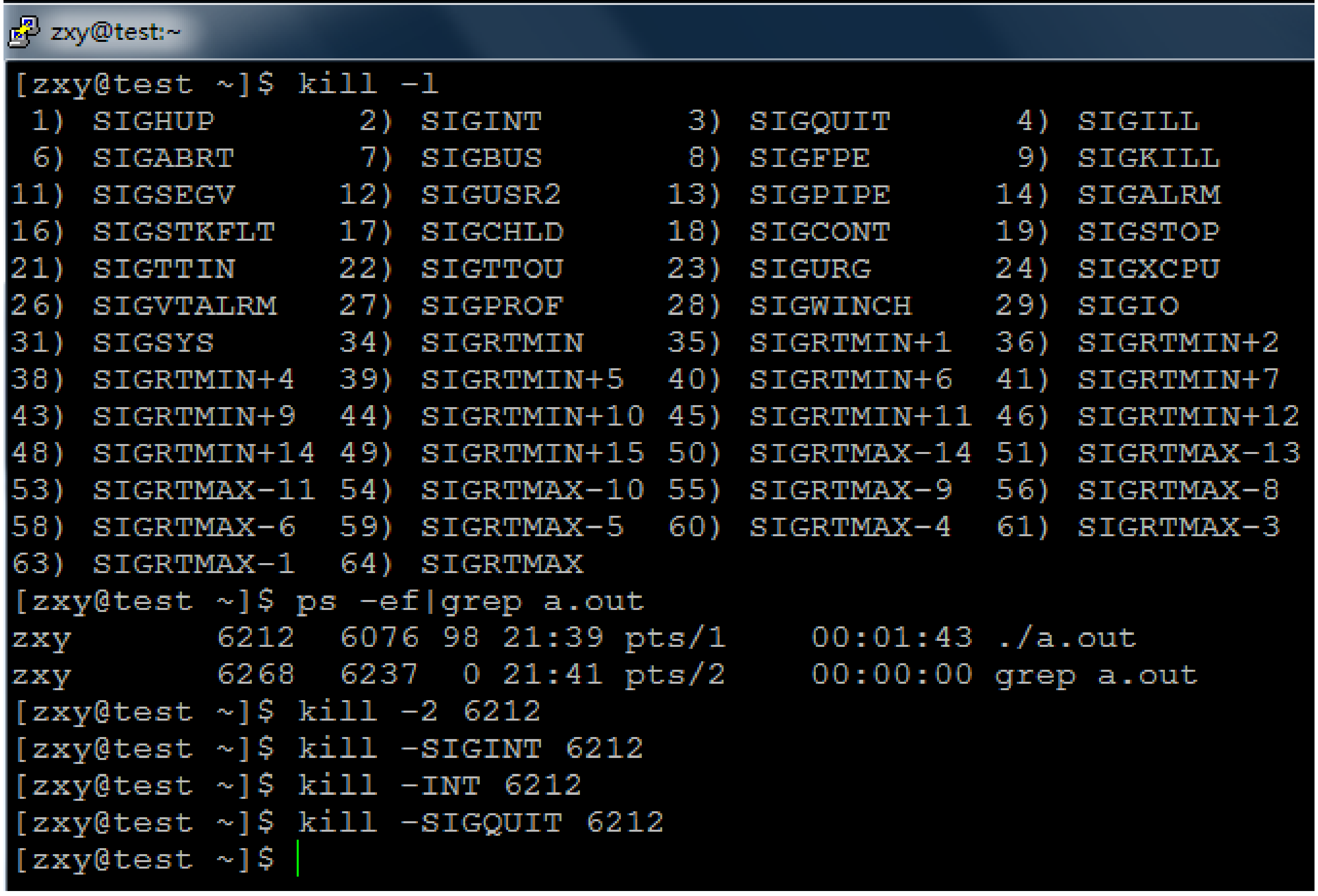
示例程序：

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h> void sig_handler( int signo);
 int main( void )
{
    printf( " mian is waiting for a signal \n " );
     if (signal(SIGINT,sig_handler) == SIG_ERR){
        perror( " signal errror " );
        exit(EXIT_FAILURE);
    } for (; ;); // 有時間讓我們發送信號return 0 ;
} void sig_handler( int signo)
{
    printf( " catch the signal SIGINT %d\n " ,signo);
}
```

結果：



```
[zxy@test unixenv_c]$ cc signal.c
[zxy@test unixenv_c]$ ./a.out
mian is waiting for a signal
^Ccatch the signal SIGINT 2
^Ccatch the signal SIGINT 2
^Ccatch the signal SIGINT 2
catch the signal SIGINT 2
catch the signal SIGINT 2
catch the signal SIGINT 2
Quit (core dumped)
[zxy@test unixenv_c]$
```



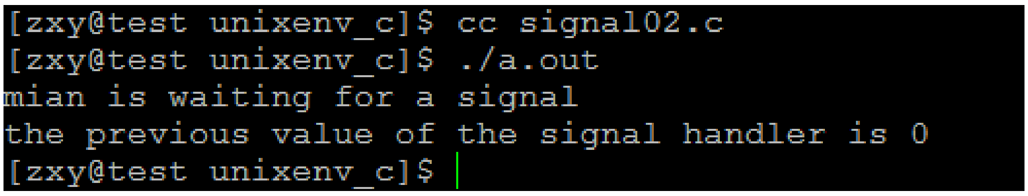
```
[zxy@test ~]$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3
63) SIGRTMAX-1  64) SIGRTMAX
[zxy@test ~]$ ps -ef|grep a.out
zxy       6212  6076 98 21:39 pts/1    00:01:43 ./a.out
zxy       6268  6237  0 21:41 pts/2    00:00:00 grep a.out
[zxy@test ~]$ kill -2 6212
[zxy@test ~]$ kill -SIGINT 6212
[zxy@test ~]$ kill -INT 6212
[zxy@test ~]$ kill -SIGQUIT 6212
[zxy@test ~]$
```

可知我們捕獲了SIGINT信號，每當我們按下ctrl+c或利用kill發送SIGINT信號時，執行我們安裝的信號處理函數，當我們按下：ctrl+\或kill –SIGQUIT pid發送SIGQUIT信號時，程序退出，那是因為進程對SIGQUIT信號的默認處理動作是退出程序。

現在我們來獲得進程的最後一次為安裝信號時所指定的處理函數：

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <sys/types.h> void sig_handler( int signo);
 int main( void )
{
    printf( " main is waiting for a signal\n " );
    __sighandler_t prehandler;
    prehandler = signal(SIGINT,sig_handler);
     if (prehandler == SIG_ERR){
        perror( " signal errror " );
        exit(EXIT_FAILURE);
    }
    printf( " the previous value of the signal handler is %d\n " ,( int )prehandler);
     // for(; ;); // 有時間讓我們發送信號return 0 ;
} void sig_handler( int signo)
{
    printf( " catch the signal SIGINT %d\n " ,signo);
}
```

結果：



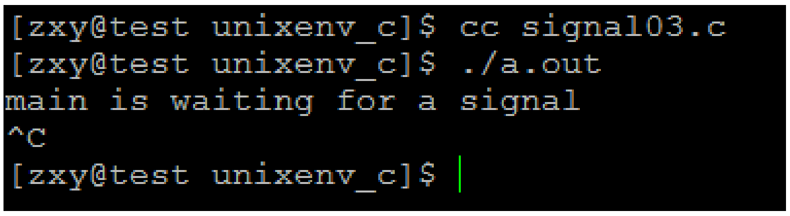
```
[zxy@test unixenv_c]$ cc signal02.c
[zxy@test unixenv_c]$ ./a.out
mian is waiting for a signal
the previous value of the signal handler is 0
[zxy@test unixenv_c]$
```

為0，由前面的宏定義：#define SIG_DFL ((__sighandler_t) 0)，可知處理動作為SIG_DFL，而SIGINT默認的處理動作就是終止進程

示例：

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <sys/types.h> void sig_handler( int signo);
 int main( void )
{
    printf( " main is waiting for a signal\n " );
    __sighandler_t prehandler;
    prehandler = signal(SIGINT,SIG_DFL);
     if (prehandler == SIG_ERR){
        perror( " signal errror " );
        exit(EXIT_FAILURE);
    } for ( ; ;); // 有時間讓我們發送信號return 0 ;
}
```

結果：



```
[zxy@test unixenv_c]$ cc signal03.c
[zxy@test unixenv_c]$ ./a.out
main is waiting for a signal
^C
[zxy@test unixenv_c]$ 
```

當按下ctrl+c時發送SIGINT信號給進程，然後進程終止

三，kill()發送信號

發送信號的主要函數有：kill()、raise()、 sigqueue()、alarm()、setitimer()以及abort()。

這裡我們先將kill函數使用：

#include <sys/types.h>

#include <signal.h>

int kill(pid_t pid,int signo)

該系統調用可以用來向任何進程或進程組發送任何信號。參數pid的值為信號的接收進程

- pid>0 進程ID為pid的進程
- pid=0 同一個進程組的進程
- pid<0 pid!=-1 進程組ID為-pid的所有進程
- pid=-1 除發送給每一個調用進程有權限發送的進程除自身及1（init）進程外

Sinno是信號值，當為0時（即空信號），實際不發送任何信號，但照常進行錯誤檢查，因此，可用於檢查目標進程是否存在，以及當前進程是否具有向目標發送信號的權限（root權限的進程可以向任何進程發送信號，非root權限的進程只能向屬於同一個session或者同一個用戶的進程發送信號）。

Kill()最常用於pid>0時的信號發送。該調用執行成功時，返回值為0；錯誤時，返回-1，並設置相應的錯誤代碼errno。下面是一些可能返回的錯誤代碼：

EINVAL：指定的信號sig無效。

ESRCH：參數pid指定的進程或進程組不存在。注意，在進程表項中存在的進程，可能是一個還沒有被wait收回，但已經終止執行的僵死進程。

EPERM： 進程沒有權力將這個信號發送到指定接收信號的進程。因為，一個進程被允許將信號發送到進程pid時，必須擁有root權力，或者是發出調用的進程的UID 或EUID與指定接收的進程的UID或保存用戶ID（savedset-user-ID）相同。如果參數pid小於-1，即該信號發送給一個組，則該錯誤表示組中有成員進程不能接收該信號。

man幫助說明：

DESCRIPTION
    The kill() system call can be used to send any signal to any process group or process.

    If pid is positive, then signal sig is sent to the process with the ID specified by pid.

    If pid equals 0, then sig is sent to every process in the process group of the calling process.

If pid equals -1, then sig is sent to every process for which the call-
ing process has permission to send signals, except for process 1
(init), but see below.

If pid is less than -1, then sig is sent to every process in the pro-
cess group whose ID is -pid.

If sig is 0, then no signal is sent, but error checking is still per-
formed; this can be used to check for the existence of a process ID or
process group ID.

For a process to have permission to send a signal it must either be
privileged (under Linux: have the CAP_KILL capability), or the real or
effective user ID of the sending process must equal the real or saved
set-user-ID of the target process. In the case of SIGCONT it suffices
when the sending and receiving processes belong to the same session.

RETURN VALUE

On success (at least one signal was sent), zero is returned. On error,
-1 is returned, and errno is set appropriately.

ERRORS

EINVAL An invalid signal was specified.

EPERM The process does not have permission to send the signal to any
of the target processes.

ESRCH The pid or process group does not exist. Note that an existing
process might be a zombie, a process which already committed
termination, but has not yet been wait(2)ed for.

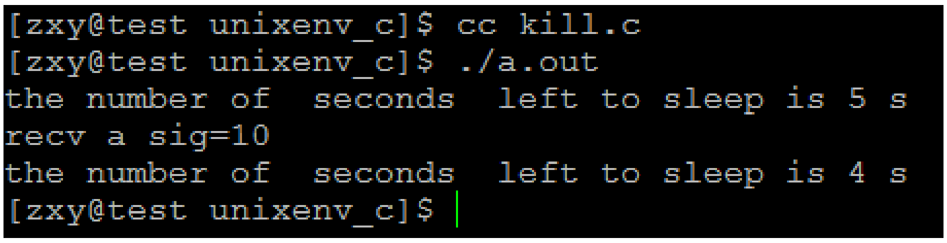
示例程序：

```
#include <unistd.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <fcntl.h>

#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include < string .h>
#include <signal.h> #define ERR_EXIT(m) \
    do \
   { \
       perror(m); \
       exit(EXIT_FAILURE); \
   } while ( 0 ) void handler( int sig);
 int main( int argc, char * argv[])
{ if (signal(SIGUSR1, handler) == SIG_ERR)
       ERR_EXIT( " signal error " );
   pid_t pid = fork() ;
    if (pid == - 1 )
       ERR_EXIT( " fork error " ); if (pid == 0 )
   {
       sleep( 1 );
```

```
        kill(getppid(), SIGUSR1);
        exit(EXIT_SUCCESS);
    } int n = 5 ;
     do
    {
        printf( " the number of seconds left to sleep is %ds\n " ,n);
        n = sleep(n);
    } while (n > 0 );
     return 0 ;
} void handler( int sig)
{
    printf( " recv a sig=%d\n " , sig);
}
```

結果：



```
[zxy@test unixenv_c]$ cc kill.c
[zxy@test unixenv_c]$ ./a.out
the number of  seconds  left to sleep is 5 s
recv a sig=10
the number of  seconds  left to sleep is 4 s
[zxy@test unixenv_c]$
```

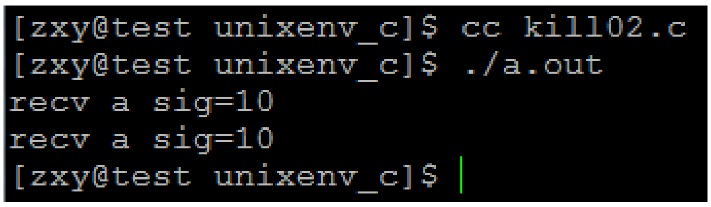
以上程序裡有子進程給父進程發送SIGUSR1信號，父進程收到信號後，睡眠被中斷，然後去執行信號處理函數，返回後繼續睡眠剩餘的時間後退出程序。

現在利用kill給與給定pid同組所有進程發送信號：

```
#include <unistd.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <fcntl.h>

#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include < string .h>
#include <signal.h> #define ERR_EXIT(m) \
     do \
    { \
        perror(m); \
        exit(EXIT_FAILURE); \
    } while ( 0 ) void handler( int sig);
 int main( int argc, char * argv[])
{ if (signal(SIGUSR1, handler) == SIG_ERR)
        ERR_EXIT( " signal error " );
    pid_t pid = fork() ;
     if (pid == - 1 )
        ERR_EXIT( " fork error " ); if (pid == 0 )
    {
        pid = getpgrp();
        kill( - pid, SIGUSR1);
         //kilpg (getpgrp(), SIGUSR1);          exit(EXIT_SUCCESS);
    } int n = 5 ;
     do
    {
        n = sleep(n);
    } while (n > 0 );
     return 0 ;
} void handler( int sig)
{
   printf( " recv a sig=%d\n " , sig);
}
```

結果：



```
[zxy@test unixenv_c]$ cc kill02.c
[zxy@test unixenv_c]$ ./a.out
recv a sig=10
recv a sig=10
[zxy@test unixenv_c]$ 
```

可知收到進行了兩次信號處理函數的執行：因為當前所屬組中只有父子兩個進程，從上可知有兩種方式給組進程發送信號：kill和killpg

四，arise函數

#include <signal.h>

int raise(int signo)

向進程本身發送信號，參數為即將發送的信號值。調用成功返回0；否則，返回-1。

man幫助說明：

DESCRIPTION

The raise() function sends a signal to the calling process or thread.
In a single-threaded program it is equivalent to

kill(getpid(), sig);

In a multithreaded program it is equivalent to

pthread_kill(pthread_self(), sig);

If the signal causes a handler to be called, raise() will only return
after the signal handler has returned.

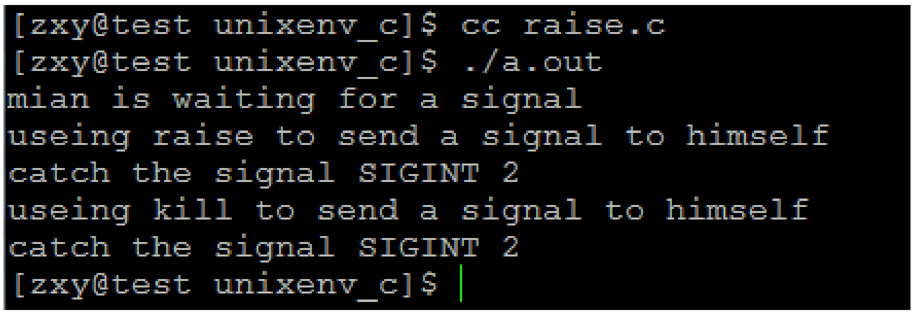
RETURN VALUE

raise() returns 0 on success, and non-zero for failure.

示例程序：

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h> void sig_handler( int signo);
 int main( void )
{
    printf( " mian is waiting for a signal \n " );
     if (signal(SIGINT,sig_handler) == SIG_ERR){
        perror( " signal errror " );
        exit(EXIT_FAILURE);
    }
    printf( " useing raise to send a signal to himself\n " );
    raise( SIGINT);
    sleep( 1 );
    printf( " useing kill to send a signal to himself\n " );
    kill(getpid(),SIGINT); return 0 ;
} void sig_handler( int signo)
{
    printf( " catch the signal SIGINT %d\n " ,signo);
}
```

結果：



```
[zxy@test unixenv_c]$ cc raise.c
[zxy@test unixenv_c]$ ./a.out
mian is waiting for a signal
useing raise to send a signal to himself
catch the signal SIGINT 2
useing kill to send a signal to himself
catch the signal SIGINT 2
[zxy@test unixenv_c]$ 
```

可知兩種方式都可以給自身發送信號。