

- Signals
  - Asynchronous
    - no specific timing
  - Architectural dependent
  - Process determines actions
    - “received” when process takes action
  - Pending signals
    - blocked signals
    - receiving process not in running state

```
$ kill -l
```

1) SIGHUP	2) SIGINT	3) SIGQUIT	4) SIGILL
5) SIGTRAP	6) SIGABRT	7) SIGEMT	8) SIGFPE
9) <b>SIGKILL</b>	10) SIGBUS	11) SIGSEGV	12) SIGSYS
13) SIGPIPE	14) SIGALRM	15) SIGTERM	16) SIGURG
17) <b>SIGSTOP</b>	18) SIGTSTP	19) SIGCONT	20) SIGCHLD
21) SIGTTIN	22) SIGTTOU	23) SIGIO	24) SIGXCPU
25) SIGXFSZ	26) SIGVTALRM	27) SIGPROF	28) SIGWINCH
29) SIGINFO	30) SIGUSR1	31) SIGUSR2	

```
$ kill -l
```

1) SIGHUP	2) SIGINT	3) SIGQUIT	4) SIGILL
5) SIGTRAP	6) SIGABRT	7) SIGBUS	8) SIGFPE
9) <b>SIGKILL</b>	10) SIGUSR1	11) SIGSEGV	12) SIGUSR2
13) SIGPIPE	14) SIGALRM	15) SIGTERM	17) SIGCHLD
18) SIGCONT	19) <b>SIGSTOP</b>	20) SIGTSTP	21) SIGTTIN
22) SIGTTOU	23) SIGURG	24) SIGXCPU	25) SIGXFSZ
26) SIGVTALRM	27) SIGPROF	28) SIGWINCH	29) SIGIO
30) SIGPWR	31) SIGSYS	33) <i>SIGRTMIN</i>	34) <i>SIGRTMIN+1</i>
35) <i>SIGRTMIN+2</i>	36) <i>SIGRTMIN+3</i>	37) <i>SIGRTMIN+4</i>	38) <i>SIGRTMIN+5</i>
39) <i>SIGRTMIN+6</i>	40) <i>SIGRTMIN+7</i>	41) <i>SIGRTMIN+8</i>	42) <i>SIGRTMIN+9</i>
43) <i>SIGRTMIN+10</i>	44) <i>SIGRTMIN+11</i>	45) <i>SIGRTMIN+12</i>	46) <i>SIGRTMIN+13</i>
47) <i>SIGRTMIN+14</i>	48) <i>SIGRTMIN+15</i>	49) <i>SIGRTMAX-15</i>	50) <i>SIGRTMAX-14</i>
51) <i>SIGRTMAX-13</i>	52) <i>SIGRTMAX-12</i>	53) <i>SIGRTMAX-11</i>	54) <i>SIGRTMAX-10</i>
55) <i>SIGRTMAX-9</i>	56) <i>SIGRTMAX-8</i>	57) <i>SIGRTMAX-7</i>	58) <i>SIGRTMAX-6</i>
59) <i>SIGRTMAX-5</i>	60) <i>SIGRTMAX-4</i>	61) <i>SIGRTMAX-3</i>	62) <i>SIGRTMAX-2</i>
63) <i>SIGRTMAX-1</i>	64) <i>SIGRTMAX</i>		

- Actions
  - Default action
  - Ignore signal
  - Catch signal

- Default action
  - terminate
    - “performs” exit system call
  - core dump
    - produce a core image and terminate
  - stop
    - suspend
  - ignore
    - disregard the signal

- Ignore
  - ignore the signal
    - except
      - SIGKILL ( 9 )
      - SIGSTOP ( 17, 19, 23 )
  - If signal is currently blocked
    - discarded

- Catch signal
  - signal catcher (handler)
    - supplied by user
    - function to be executed upon receiving signal
    - resume normal execution

- signal generation
  - Kernel
    - Hardware conditions
      - SIGSEGV  
addressing violation
      - SIGFPE  
division by zero
    - Software conditions
      - SIGIO  
I/O
      - expired timer

- User
  - Keyboard

SIGINT ( 2 )

<C>

SIGQUIT ( 3 ) (produce a core dump )

<\\>

```
$ stty -a
speed 9600 baud; rows 57; columns 169; line = 0;
intr = ^C; quit = ^\; erase = ^?; eof = ^D;
eol = M-^?; eol2 = M-^?; start = ^Q; stop = ^S; susp = ^Z;
rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O;
min = 1; time = 0;
...
```



- kill command
  - same EUID
  - kill [ -signal] pid ...
    - default signal SIGTERM ( 15 )

- By another process

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

pid	process receiving signal
>0	process whose process ID is pid
0	all processes in process group of sender.
-1	<i>not superuser</i> : all processes whose real ID is the same as effective ID of sender <i>superuser</i> : all processes excluding special processes
<-1	all the processes whose process group is absolute value of (-pid) (  -pid  )

sig:

any symbolic (or equivalent integer)

sig == 0

perform error check on PID

will not send signal

Linux

telinit to send signal to init

- alarm system call
  - `unsigned int alarm ( unsigned int seconds );`
    - sets a timer
      - timer expires
        - SIGALRM
      - zero resets timer
    - `fork()` processes
      - alarm reset: `alarm(0)`
    - `exec ( )` processes
      - inherit alarm remaining time
  - cannot be stacked
    - multiple calls resets alarm

- `pause`
  - `int pause ( void );`
  - suspends a process until a signal that is not ignored is received
  - returns:
    - -1 if received signal does not cause termination

- Signal Management
  - Ignoring signal
  - Catching signal
- System calls
  - `signal`
  - `sigaction`

**Table 4.19** Summary of the `signal` System Call.

Include File(s)	<signal.h>		Manual Section	<b>2</b>
Summary	<pre>void (*signal(int signum,                void (*sighandler)(int)))(int);</pre>			
Return	Success	Failure	Sets <code>errno</code>	
	Signal's previous disposition	SIG_ERR (defined as -1)	Yes	

**Table 4.20** Summary of the `sigaction` System Call.

Include File(s)	<signal.h>		Manual Section	<b>2</b>
Summary	<pre>int sigaction(int signum, const                struct sigaction *act,                struct sigaction *oldact);</pre>			
Return	Success	Failure	Sets <code>errno</code>	
	0	-1	Yes	

- signal arguments

- `void (*signal(int signum,void(*sighandler)(int)))(int );`
  - an integer `signum` values
    - cannot be `SIGKILL` or `SIGSTOP`
  - a pointer to a function
  - returns a pointer to a function which returns nothing (`void`)
  - `last (int)`
    - referenced function has an integer argument
    - filled by the system with signal number



- signal system call
  - integer or symbolic name
    - except SIGKILL or SIGSTOP
  - address of signal catcher
    - User defined function
    - SIG\_DFL
      - default action
    - SIG\_IGN
      - ignore signal

```
if (signal(SIGHUP, SIG_IGN) == SIG_ERR) {  
    perror("SIGHUP");  
    return 3;  
}
```

**nohup** is a Unix command that is used to run another command while suppressing the action of the HUP (hangup) signal, *enabling the command to keep running after the user who issues the command has logged out*. It is most often used to run commands in the *background as daemons*. Output that would normally go to the terminal goes to a file called `nohup.out` if it has not already been redirected.

**Program 4.4** Pseudo nohup—ignoring a signal.

```

File : p4.4.cxx
|      /* Using the signal system call to ignore a hangup signal
|      */
|      #include <iostream>
+      #include <cstdio>
|      #include <cstdlib>
|      #include <signal.h>
|      #include <fcntl.h>
|      #include <unistd.h>
|      using namespace std;
10     const char *file_out = "nohup.out";
|     int
|     main(int argc, char *argv[]){
|         int      new_stdout;
|         if (argc < 2) {
+             cerr << "Usage: " << *argv << " command [arguments]" << endl;
|             return 1;
|         }
|         if (isatty( 1 )) {
|             cerr << "Sending output to " << file_out << endl;
20         close( 1 );
|         if ((new_stdout = open(file_out, O_WRONLY | O_CREAT |
|                                 O_APPEND, 0644)) == -1) {
|             perror(file_out);
|             return 2;
+         }
|         }
|         if (signal(SIGHUP, SIG_IGN) == SIG_ERR) {
|             perror("SIGHUP");
|             return 3;
30     }
|     ++argv;
|     execvp(*argv, argv);
|     perror(*argv);           // Should not get here unless
|     return 4;               // the exec call fails.
+     }

```

- Signal catcher
  - prior to calling function
    - Signal is reset to default action
      - except for SIGKILL, SIGPWR and SIGTRAP
- signals may be lost
  - consecutive signals
  - reset signal handler within signal catcher

```
/* Catching a signal */
#include <dostream>
#include <cstdlib> #include <cstdio>
#include <signal. h>
#include <unistd.h>
using namespace std;
int main( )
( void signal_catcher (int) ;
if (signal (SIGINT , signal_catcher) == SIG_ERR) {
    perror ( "SIGINT" ) ;
    return 1;
}
if (signal (SIGQUIT , signal_catcher) == SIG_ERR) {
    perror (."SIGQUIT" ) ;
    return 2;
}
for (int i=0; ; ++ i) {          // forever
    cout « i « endl;           // display number
    sleep (1) ;
}
return 0;
void signal_catcher(int the_sig) {
    signal (the_sig, signal_catcher); // reset immediately
    cout « endl « "Signal « the_sig « " received.. « endl ;
    if ( the_sig == SIGQUIT(
        exit(3);
}
```

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

- **sigaction**

- Arguments

- signal
    - sigaction structure
      - new and previous actions
    - sa\_handler
    - sa\_mask
      - signals to be blocked while handler is running
      - bit representation
      - signal that triggered the handler is blocked
    - sa\_flag
      - modify behavior of signal handler

**signal catcher remains active**

- **sa\_flag**

**Table 4.22** sa\_flags Constants.

Flag	Action
SA_NOCLDSTOP	If the signal is SIGCHLD, then the calling process will not receive a SIGCHLD signal when its child processes exit.
SA_ONESHOT or SA_RESETHAND	Restore the default action after the signal handler has been called once (similar to the default of the <b>signal</b> call).
SA_RESTART	Use BSD signal semantics (certain interrupted system calls are restarted after the signal has been caught).
SA_NOMASK or SA_NODEFER	Undo the default whereby the signal triggering the handler is automatically blocked.
SA_SIGINFO	The signal handler has three arguments—use sa_sigaction, not sa_handler.

```
| #include <signal.h>
| #include <unistd.h>
| using namespace std;
10 int
| main( ) {
|     void    signal_catcher(int);
|     struct sigaction new_action;
|     new_action.sa_handler = signal_catcher;
+    new_action.sa_flags    = 0;
|
|     if (sigaction(SIGINT, &new_action, NULL) == -1) {
|         perror("SIGINT");
|         return 1;
20     }
|     if (sigaction(SIGQUIT, &new_action, NULL) == -1) {
|         perror("SIGQUIT");
|         return 2;
|     }
+    for (int i=0; ; ++i) { // Forever ...
|        cout << i << endl; // display a number
|        sleep(1);
|    }
|    return 0;
30 }
| void
| signal_catcher(int the_sig){
|     cout << endl << "Signal " << the_sig << " received." << endl;
|     if (the_sig == SIGQUIT)
+     exit(3);
| }
```

A sigaction structure is allocated.

The signal catching function is assigned and the sa\_flags member set to 0.

A new action is associated with each signal.



- Signal mask related system calls
  - sigprocmask
  - sigpending
  - sigsuspend

**Table 4.23** Summary of the sigprocmask, sigpending, and sigsuspend System Call.

Include File(s)	<unistd.h>		Manual Section	<b>2</b>
Summary	<pre>int sigprocmask (int how, const sigset_t *set,                  sigset_t *oldset); int sigpending(sigset_t *set); int sigsuspend(const sigset_t *mask);;</pre>			
Return	Success	Failure	Sets errno	
	0	-1	Yes	



- `sigprocmask`
  - how
    - `SIG_BLOCK`
      - block the signals specified by the union of the current set of signals with those specified by the set argument
    - `SIG_UNBLOCK`
      - unblock the signals specified by the set argument
    - `SIG_SETMASK`
      - block the signals specified by the set argument
  - `oldset`
    - previous value of signal mask is saved

- `sigsuspend`
  - suspend a process
  - replaces current signal mask with one passed as argument
  - until a signal is delivered whose action is to execute a signal catching function or terminate a process.
- `sigpending`
  - a mask of the signals pending for delivery to the calling process in the location indicated by set

- library functions
  - `sigempty`
    - clear signal mask
  - `sigaddset`
    - add signals to signal mask