Primitive Communications
Signals

- Signals

- Signals
  - sent by kernel or user
  - notification that an event has occurred
  - 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  (BSD )
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGEMT       8) SIGFPE
 9) SIGKILL     10) SIGBUS      11) SIGSEGV     12) SIGSYS
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGURG
17) SIGSTOP     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  ( Linux )
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     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) SIGRTMIN    34) SIGRTMIN+1
35) SIGRTMIN+2  36) SIGRTMIN+3  37) SIGRTMIN+4  38) SIGRTMIN+5
39) SIGRTMIN+6  40) SIGRTMIN+7  41) SIGRTMIN+8  42) SIGRTMIN+9
43) SIGRTMIN+10 44) SIGRTMIN+11 45) SIGRTMIN+12 46) SIGRTMIN+13
47) SIGRTMIN+14 48) SIGRTMIN+15 49) SIGRTMAX-15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Table 4.13. Signal Definitions.

Symbolic Name	Def	Value	Action	Description
SIGABRT	P	6	C	Abort signal from abort.
SIGALRM	P	14	A	Timer signal from alarm.
SIGBUS	S	10,7,10	C	Bus error (bad memory access).
SIGCHLD	P	20,17,18	B	Sent to parent when child is stopped or terminated.
SIGCLD	O	-,-,18	B	A synonym for SIGCHLD.
SIGCONT	P	19,18,25	B	Resume if process is stopped.
SIGEMT	O	7,-,7	C	Emulation trap.
SIGFPE	P	8	C	Floating-point exception.
SIGHUP	P	1	A	A hangup was detected on the controlling terminal or the controlling process has died.
SIGILL	P	4	C	Illegal instruction.
SIGINFO	O	29,-,-		A synonym for SIGPWR.
SIGINT	P	2	A	Interrupt from keyboard.
SIGIO	O	23,29,22	A	I/O now possible.
SIGIOT	O	6	C	IOT trap—equivalent to SIGABRT.
SIGKILL	P	9	A,E,F	Kill signal—force process termination.
SIGLOST	O	-,-,-	A	File lock lost.
SIGPIPE	P	13	A	Broken pipe; write to pipe with no readers.
SIGPOLL	S	23	A	A pollable event has occurred—synonymous with SIGIO (also 23).
SIGPROF	S	27,27,29	A	Profiling timer expired.
SIGPWR	O	29,30,19	A	Power supply failure.
SIGQUIT	P	3	C	Quit from keyboard.
SIGSEGV	P	11	C	Invalid memory reference (segmentation violation).
SIGSTKFLT	O	-,16,-	A	Coprocessor stack error.
SIGSTOP	P	17,19,23	D,E,F	Stop process—not from tty.
SIGSYS	S	12,-,12	C	Bad argument to system call.
SIGTERM	P	15	A	Termination signal from kill.
SIGTRAP	S	5	C	Trace/breakpoint trap for debugging.
SIGTSTP	P	18,20,24	D	Stop typed at a tty.
SIGTTIN	P	21,21,26	D	Background process needs input.
SIGTTOU	P	22,22,27	D	Background process needs to output.
SIGUNUSED	O	-,31,-	A	Unused signal (will be SIGSYS).
SIGURG	S	16,23,21	B	Urgent condition on I/O channel (socket).
SIGUSR1	P	30,10,16	A	User-defined signal 1.
SIGUSR2	P	31,12,17	A	User-defined signal 2.
SIGVTALRM	S	26,26,28	A	Virtual alarm clock.
SIGWINCH	O	28,28,20	B	Window resize signal.
SIGXCPU	S	24,24,30	C	CPU time limit exceeded.
SIGXFSZ	S	25,25,31	C	File size limit exceeded.

```
Action:
A: terminate
B:ignore
C:terminate/dump
D:stop/suspend
E:cannot be caught caught
F:cannot be ignored
```

- 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
  - user on a terminal
  - another process
  - alarm system call

- 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)
- other process
  - `kill()` system call

- By another process

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

pid	process(es) receiving signal
>0	process whose process ID is pid
0	all processes in process group of sender.
-1	*not superuser*: all processes whose real ID is the same as effective ID of sender *superuser*: 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

- by 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
  - Default

  - System calls
    - `signal`
    - `sigaction`

Table 4.19 Summary of the `signal` System Call.

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

Table 4.20 Summary of the `sigaction` System Call.

Include File(s)	`<signal.h>`	Manual Section	2
Summary	`int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);`		
Return	Success: 0	Failure: −1	Sets `errno`: 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 …
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
    printf("%d\n",i);;         // display number
    sleep (1) ;
 }
return 0;
void signal_catcher(int the_sig) {
    signal (the_sig, signal_catcher);  //  reset immediately
    printf("Signal %d\n received",the_sig)'
    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 SIGCHILD, then the calling process will not receive a SIGCHILD 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 **`signal`** 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;
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;
  }
  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;
}
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 action stays active

- 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	2

Summary	`int sigprocmask (int how, const sigset_t *set, sigset_t *oldset);`
	`int sigpending(sigset_t *set);`
	`int sigsuspend(const sigset_t *mask);;`		

Return	Success: 0	Failure: −1	Sets `errno`: 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