

Chapter 4 Signals - Notes

4.2 Introduction

Signals: used to emit notifications for processes to take action in response to often unpredictable events. May be caused from within process itself, or external events such as other processes.

Many signals fatal, resulting in process termination. Death can sometimes be averted if program designers decide to handle (subvert) certain termination signals.

Many signals more benign, just informative or request other kinds of actions. Possible to send signals (including those that induce termination) from command line using **kill**, **killall**, **pkill**.

4.3 Learning Objectives:

- Explain what signals are and how they are used.
- Know the available signals and types of signals available in Linux.
- Use kill, killall, and pkill to send signals from the command line.

4.4 What are Signals?

Signals: one of oldest methods of **Inter-Process Communication (IPC)**, used to notify processes about **asynchronous** events (or exceptions).

By asynchronous, the signal-receiving process may:

- Not expect event to occur
- Expect event, but not know when it is most likely to occur

Example: user decides to terminate running program. Could send signal to process through kernel to interrupt and kill process.

Two paths by which signals sent to process:

- From kernel to user process, as result of exception or programming error
- From user process (using system call) to the kernel which will then send to user process. Process sending signal can actually be same as one receiving signal

Signals only sent between processes owned by same user or from process owned by superuser to any process.

When process receives signal, what it does depends on way program is written. Can take specific actions (coded into program) to **handle** signal or it can just respond according to system defaults. Two signals (**SIGKILL** and **SIGSTOP**) cannot be handled and always terminate program.

4.5 Types of Signals

Number of different types of signals, particular signal dispatched indicates type of event (or exception) occurred. Generally, signals used to handle:

1. Exceptions detected by hardware (eg. illegal memory reference)
2. Exceptions generated by environment (eg. premature death of process from user's terminal)

TO see list of signals in Linux, along with numbers, do `kill -l`, as reflected in screenshot.

```
student@openSUSE:~
File Edit View Search Terminal Help
student@openSUSE:~> kill -l
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
16) SIGSTKFLT  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     34) SIGRTMIN   35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+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
student@openSUSE:~>
```

Signals from `SIGRTMIN` on termed **real-time signals**, relatively recent addition. No predefined purpose, differ in some important ways from normal signals. Can be queued up and handled in **FIFO** (F irst I n F irst O ut) order.

Meaning attached to signal type indicates event that caused signal to be sent. While users can explicitly send any signal type to their processes, meaning attached may not longer be implied by signal number or type, can be used in any way that the process desires.

Further documentation: `man 7 signal`.

Available Signals for the x86 Platform

| Signal | Value | Default Action | POSIX? | Meaning |
|----------------|-------|----------------|--------|---|
| SIGHUP | 1 | Terminate | Yes | Hangup detected on controlling terminal or death of controlling process |
| SIGINT | 2 | Terminate | Yes | Interrupt from keyboard |
| SIGQUIT | 3 | Core dump | Yes | Quit from keyboard |
| SIGILL | 4 | Core dump | Yes | Illegal instruction |
| SIGTRAP | 5 | Core dump | No | Trace/breakpoint trap for debugging |
| SIGABRT/SIGIOT | 6 | Core dump | Yes | Abnormal termination |
| SIGBUS | 7 | Core dump | Yes | Bus error |
| SIGFPE | 8 | Core dump | Yes | Floating point exception |
| SIGKILL | 9 | Terminate | Yes | Kill signal (cannot be caught or ignored) |
| SIGUSR1 | 10 | Terminate | Yes | User-defined signal 1 |
| SIGSEGV | 11 | Core dump | Yes | Invalid memory reference |

| | | | | |
|-------------------------|----|-----------|-----|--|
| SIGUSR2 | 12 | Terminate | Yes | User-defined signal 2 |
| SIGPIPE | 13 | Terminate | Yes | Broken pipe: write to pipe with no readers |
| SIGALRM | 14 | Terminate | Yes | Timer signal from alarm |
| SIGTERM | 15 | Terminate | Yes | Process termination |
| SIGSTKFLT | 16 | Terminate | No | Stack fault on math co-processor |
| SIGCHLD | 17 | Ignore | Yes | Child stopped or terminated |
| SIGCONT | 18 | Continue | Yes | Continue if stopped |
| SIGSTOP | 19 | Stop | Yes | Stop process (cannot be caught or ignored) |
| SIGTSTP | 20 | Stop | Yes | Stop types at tty |
| SIGTTIN | 21 | Stop | Yes | Background process requires tty input |
| SIGTTOU | 22 | Stop | Yes | Background process requires tty output |
| SIGURG | 23 | Ignore | No | Urgent condition on socket (4.2 BSD) |
| SIGXCPU | 24 | Core dump | Yes | CPU time limit exceeded (4.2 BSD) |
| SIGXFSZ | 25 | Core dump | Yes | File size limit exceeded (4.2 BSD) |
| SIGVTALRM | 26 | Terminate | No | Virtual alarm clock (4.2 BSD) |
| SIGPROF | 27 | Terminate | No | Profile alarm clock (4.2 BSD) |
| SIGWINCH | 28 | Ignore | No | Window resize signal (4.3 BSD, Sun) |
| SIGIO/SIGPOLL | 29 | Terminate | No | I/O now possible (4.2BSD) (System V) |
| SIGPWR | 30 | Terminate | No | Power Failure (System V) |
| SIGSYS/SIGUNUSED | 31 | Terminate | No | Bad System Called. Unused signal. |

4.6 kill

Process cannot send signal directly to another process, must ask kernel to send signal by executing **system call**. Users (including superuser) can send signals to other processes from command line or scripts using **kill**:

```
$ kill 1991
$ kill -9 1991
$ kill -SIGKILL 1991
```

where user sending signal to process with **PID = 1991**. If signal number not given (as in first example), default to send **SIGTERM (15)**, terminate signal that can be handled. Program can take elusive action or clean up after itself, rather than die immediately. If this signal ignored, user can usually send **SIGKILL (9)** (cannot be ignored), to terminate with extreme prejudice.

Name **kill** -> really bad name, misnomer that survives for historical reasons. Although often used to kill (terminate) processes, command's real function: send any and all signals to processes, even totally benign informative ones.

4.7 killall and pkill

killall: kills all processes with given name, assuming user has sufficient privilege. Uses command name rather than process ID:

```
$ killall bash
$ killall -9 bash
$ killall -SIGKILL bash
```

pkill sends signal to process using selection criteria:

```
$ pkill [-signal] [options] [pattern]
```

For example:

```
$ pkill -u libby foobar
```

will kill all of **libby**'s processes with a name of **foobar**.

Another example:

```
$ pkill -HUP rsyslogd
```

makes **rsyslog** re-read its configuration file.

##

[Back to top](#)

[Previous Chapter](#) - [Table of Contents](#) - [Next Chapter](#)