A signal is a software generated interrupt that is sent to a process by the OS because of when user press ctrl-c or another process tell something to this process.  
There are fix set of signals that can be sent to a process. signal are identified by integers.  
Signal number have symbolic names. For example **SIGCHLD** is number of the signal sent to the parent process when child terminates.

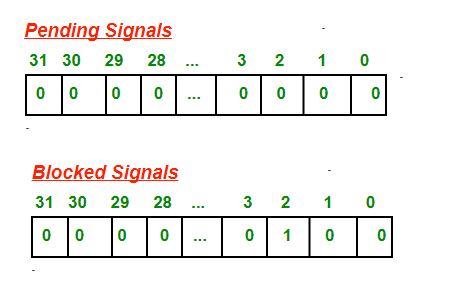
**Examples:**

|  |
| --- |
| #define SIGHUP 1 /\* Hangup the process \*/  #define SIGINT 2 /\* Interrupt the process \*/  #define SIGQUIT 3 /\* Quit the process \*/  #define SIGILL 4 /\* Illegal instruction. \*/  #define SIGTRAP 5 /\* Trace trap. \*/  #define SIGABRT 6 /\* Abort. \*/ |

**OS Structures for Signals**

* For each process, the operating system maintains 2 integers with the bits corresponding to a signal numbers.
* The two integers keep track of: **pending signals and blocked signals**
* With 32 bit integers, up to 32 different signals can be represented.

**Example :**  
In the example below, the SIGINT ( = 2) signal is blocked and no signals are pending.



A signal is sent to a process setting the corresponding bit in the pending signals integer for the process. Each time the OS selects a process to be run on a processor, the pending and blocked integers are checked. If no signals are pending, the process is restarted normally and continues executing at its next instruction.

If 1 or more signals are pending, but each one is blocked, the process is also restarted normally but with the signals still marked as pending. If 1 or more signals are pending and NOT blocked, the OS executes the routines in the process’s code to handle the signals.

**Default Signal Handlers**

There are several default signal handler routines. Each signal is associated with one of these default handler routine. The different default handler routines typically have one of the following actions:

* Ign: Ignore the signal; i.e., do nothing, just return
* Term: terminate the process
* Cont: unblock a stopped process
* Stop: block the process

|  |  |
| --- | --- |
| |  | | --- | | // CPP program to illustrate  // default Signal Handler  #include<stdio.h>  #include<signal.h>  int main()  {  signal(SIGINT, handle\_sigint);  while (1)  {  printf(“hello world\n”);  sleep(1);  }  return 0;  } | |

Output: Print hello world infinite times. If user presses ctrl-c to terminate the process because of **SIGINT** signal sent and its default handler to terminate the process.

|  |
| --- |
| hello world  hello world  hello world  terminated |

**User Defined Signal Handlers**

A process can replace the default signal handler for almost all signals (but not SIGKILL) by its user’s own handler function.  
A signal handler function can have any name, but must have return type void and have one int parameter.  
**Example:** you might choose the name sigchld\_handler for a signal handler for the **SIGCHLD** signal (termination of a child process). Then the declaration would be:

|  |
| --- |
| **void sigchld\_handler(int sig);** |

When a signal handler executes, the parameter passed to it is the number of the signal. A programmer can use the same signal handler function to handle several signals. In this case the handler would need to check the parameter to see which signal was sent. On the other hand, if a signal handler function only handles one signal, it isn’t necessary to bother examining the parameter since it will always be that signal number.

|  |
| --- |
| // CPP program to illustrate  // User-defined Signal Handler  #include<stdio.h>  #include<signal.h>  // Handler for SIGINT, caused by  // Ctrl-C at keyboard  void handle\_sigint(int sig)  {  printf("Caught signal %d\n", sig);  }  int main()  {  signal(SIGINT, handle\_sigint);  while (1) ;  return 0;  } |

Output:

|  |
| --- |
| ^CCaught signal 2 // when user presses ctrl-c  ^CCaught signal 2 |

**Sending signals via kill()**

We can send a signal using kill() to the process.

|  |
| --- |
| **int kill(pid\_t pid, int signal);**  **pid:** id of destination process  **signal:** the type of signal to send  **Return value:** 0 if signal was sent successfully |

Example:

|  |
| --- |
| pid\_t iPid = getpid(); /\* Process gets its id.\*/  kill(iPid, SIGINT); /\* Process sends itself a **SIGINT** signal  (commits suicide?)(because of **SIGINT**  signal default handler is terminate the process) \*/ |