After a signal has been handled, whether or not the handler needs to be reinstalled is implementation-defined. UNIX implementations typically leave the signal handler installed after it's been used, but other implementations may reset the handler to `SIG_DFL`. In the latter case, the handler can reinstall itself by calling `signal` before it returns.

C99 changes the signal-handling process in a few minor ways. When a signal is raised, an implementation may choose to disable not just that signal but others as well. If a signal-handling function returns from handling a `SIGILL` or `SIGSEGV` signal (as well as a `SIGFPE` signal), the effect is undefined. C99 also adds the restriction that if a signal occurs as a result of calling the `abort` function or the `raise` function, the signal handler itself must not call `raise`.

The `raise` Function

```
int raise(int sig);
```

raise

Although signals usually arise from run-time errors or external events, it's occasionally handy for a program to cause a signal to occur. The `raise` function does just that. The argument to `raise` specifies the code for the desired signal:

```
raise(SIGABRT);    /* raises the SIGABRT signal */
```

The return value of `raise` can be used to test whether the call was successful: zero indicates success, while a nonzero value indicates failure.

PROGRAM

Testing Signals

The following program illustrates the use of signals. First, it installs a custom handler for the `SIGINT` signal (carefully saving the original handler), then calls `raise_sig` to raise that signal. Next, it installs `SIG_IGN` as the handler for the `SIGINT` signal and calls `raise_sig` again. Finally, it reinstalls the original handler for `SIGINT`, then calls `raise_sig` one last time.

tsignal.c

```
/* Tests signals */

#include <signal.h>
#include <stdio.h>

void handler(int sig);
void raise_sig(void);

int main(void)
{
  void (*orig_handler)(int);
```