7.14 Signal handling `<signal.h>`

1 The header `<signal.h>` declares a type and two functions and defines several macros, for handling various *signals* (conditions that may be reported during program execution).

2 The type defined is

```
        sig_atomic_t
```

which is the (possibly volatile-qualified) integer type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts.

3 The macros defined are

```
        SIG_DFL
        SIG_ERR
        SIG_IGN
```

which expand to constant expressions with distinct values that have type compatible with the second argument to, and the return value of, the **`signal`** function, and whose values compare unequal to the address of any declarable function; and the following, which expand to positive integer constant expressions with type **`int`** and distinct values that are the signal numbers, each corresponding to the specified condition:

SIGABRT	abnormal termination, such as is initiated by the **`abort`** function
SIGFPE	an erroneous arithmetic operation, such as zero divide or an operation resulting in overflow
SIGILL	detection of an invalid function image, such as an invalid instruction
SIGINT	receipt of an interactive attention signal
SIGSEGV	an invalid access to storage
SIGTERM	a termination request sent to the program

4 An implementation need not generate any of these signals, except as a result of explicit calls to the **`raise`** function. Additional signals and pointers to undeclarable functions, with macro definitions beginning, respectively, with the letters **`SIG`** and an uppercase letter or with **`SIG_`** and an uppercase letter,[254] may also be specified by the implementation. The complete set of signals, their semantics, and their default handling is implementation-defined; all signal numbers shall be positive.

7.14.1 Specify signal handling

7.14.1.1 The `signal` function

Synopsis

1
```
        #include <signal.h>
        void (*signal(int sig, void (*func)(int)))(int);
```

Description

2 The **`signal`** function chooses one of three ways in which receipt of the signal number `sig` is to be subsequently handled. If the value of `func` is **`SIG_DFL`**, default handling for that signal will occur. If the value of `func` is **`SIG_IGN`**, the signal will be ignored. Otherwise, `func` shall point to a function to be called when that signal occurs. An invocation of such a function because of a signal, or (recursively) of any further functions called by that invocation (other than functions in the standard library),[255] is called a *signal handler*.

[254]See "future library directions" (7.31.7). The names of the signal numbers reflect the following terms (respectively): abort, floating-point exception, illegal instruction, interrupt, segmentation violation, and termination.

[255]This includes functions called indirectly via standard library functions (e.g., a **`SIGABRT`** handler called via the **`abort`** function).

3 When a signal occurs and func points to a function, it is implementation-defined whether the equivalent of **signal**(sig, **SIG_DFL**); is executed or the implementation prevents some implementation-defined set of signals (at least including sig) from occurring until the current signal handling has completed; in the case of **SIGILL**, the implementation may alternatively define that no action is taken. Then the equivalent of (*func)(sig); is executed. If and when the function returns, if the value of sig is **SIGFPE**, **SIGILL**, **SIGSEGV**, or any other implementation-defined value corresponding to a computational exception, the behavior is undefined; otherwise the program will resume execution at the point it was interrupted.

4 If the signal occurs as the result of calling the **abort** or **raise** function, the signal handler shall not call the **raise** function.

5 If the signal occurs other than as the result of calling the **abort** or **raise** function, the behavior is undefined if the signal handler refers to any object with static or thread storage duration that is not a lock-free atomic object other than by assigning a value to an object declared as **volatile sig_atomic_t**, or the signal handler calls any function in the standard library other than

- — the **abort** function,
- — the **_Exit** function,
- — the **quick_exit** function,
- — the functions in <stdatomic.h> (except where explicitly stated otherwise) when the atomic arguments are lock-free,
- — the **atomic_is_lock_free** function with any atomic argument, or
- — the **signal** function with the first argument equal to the signal number corresponding to the signal that caused the invocation of the handler. Furthermore, if such a call to the **signal** function results in a **SIG_ERR** return, the value of **errno** is indeterminate.[256)]

6 At program startup, the equivalent of

```
      signal(sig, SIG_IGN);
```

may be executed for some signals selected in an implementation-defined manner; the equivalent of

```
      signal(sig, SIG_DFL);
```

is executed for all other signals defined by the implementation.

7 Use of this function in a multi-threaded program results in undefined behavior. The implementation shall behave as if no library function calls the **signal** function.

Returns

8 If the request can be honored, the **signal** function returns the value of func for the most recent successful call to **signal** for the specified signal sig. Otherwise, a value of **SIG_ERR** is returned and a positive value is stored in **errno**.

Forward references: the **abort** function (7.22.4.1), the **exit** function (7.22.4.4), the **_Exit** function (7.22.4.5), the **quick_exit** function (7.22.4.7).

7.14.2 Send signal

7.14.2.1 The raise function

Synopsis

1
```
      #include <signal.h>
      int raise(int sig);
```

256)If any signal is generated by an asynchronous signal handler, the behavior is undefined.

Description

2 The **`raise`** function carries out the actions described in 7.14.1.1 for the signal `sig`. If a signal handler is called, the **`raise`** function shall not return until after the signal handler does.

Returns

3 The **`raise`** function returns zero if successful, nonzero if unsuccessful.