C signal handling

From Wikipedia, the free encyclopedia

In the C Standard Library, **signal processing** defines how a program handles various signals while it executes. A signal can report some exceptional behavior within the program (*such as division by zero*), or a signal can report some asynchronous event outside the program (*such as someone striking an interactive attention key on a keyboard*).

Contents

Standard signals

The C standard defines only 6 signals. They are all defined in `signal.h` header (`csignal` header in C++):[1]

- `SIGABRT` - "abort", abnormal termination.
- `SIGFPE` - **f**loating **p**oint **e**xception.
- `SIGILL` - "illegal", invalid instruction.
- `SIGINT` - "interrupt", interactive attention request sent to the program.
- `SIGSEGV` - "**seg**mentation **v**iolation", invalid memory access.
- `SIGTERM` - "terminate", termination request sent to the program.

Additional signals may be specified in the `signal.h` header by the implementation. For example, Unix and Unix-like operating systems (such as Linux) define more than 15 additional signals; see Unix signal.[2]

Handling

A signal can be generated by calling `raise()` or `kill()` system calls. `raise()` sends a signal to the current process, `kill()` sends a signal to a specific process.

A signal handler can be specified for all but two signals (SIGKILL and SIGSTOP cannot be caught, blocked or ignored). A signal handler is a function which is called by the target environment when the corresponding signal occurs. The target environment suspends execution of the program until the signal handler returns or calls `longjmp()`. For maximum portability, an asynchronous signal handler should only:

- make successful calls to the function `signal()`
- assign values to objects of type volatile `sig_atomic_t`
- return control to its caller

If the signal reports an error within the program (and the signal is not asynchronous), the signal handler can terminate by calling `abort()`, `exit()`, or `longjmp()`.

Functions

Function	Description
raise (http://en.cppreference.com/w/c/program/raise)	artificially raises a signal
signal (http://en.cppreference.com/w/c/program/signal)	sets the action taken when the program receives a specific signal

Example usage

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

static void catch_function(int signo) {
    puts("Interactive attention signal caught.");
}

int main(void) {
    if (signal(SIGINT, catch_function) == SIG_ERR) {
        fputs("An error occurred while setting a signal handler.\n", stderr);
        return EXIT_FAILURE;
    }
    puts("Raising the interactive attention signal.");
    if (raise(SIGINT) != 0) {
        fputs("Error raising the signal.\n", stderr);
        return EXIT_FAILURE;
    }
    puts("Exiting.");
    return EXIT_SUCCESS;
    // exiting after raising signal
}
```

See also

- Unix signal

References

1. *ISO/IEC 9899:1999 specification* (PDF). p. 258, § 7.14 *Signal handling*.
2. "The Open Group Base Specifications Issue 6 - signal.h - signals". Retrieved 10 January 2012.