NAME

sigfpe – signal handling for specific SIGFPE codes

SYNOPSIS

#include <floatingpoint.h>

#include <siginfo.h>

sigfpe_handler_type sigfpe(sigfpe_code_type *code***, sigfpe_handler_type** *hdl***);**

MT-LEVEL

Safe

DESCRIPTION

This function allows signal handling to be specified for particular **SIGFPE** codes. A call to **sigfpe( )** defines a new handler *hdl* for a particular **SIGFPE** *code* and returns the old handler as the value of the function **sigfpe( ).** Normally handlers are specified as pointers to functions; the special cases **SIGFPE_IGNORE**, **SIGFPE_ABORT**, and **SIGFPE_DEFAULT** allow ignoring, dumping core using **abort**(3C), or default handling respectively. Default handling is to dump core using **abort**(3C).

code is usually one of the five IEEE 754-related **SIGFPE** codes:

FPE_FLTRES	fp_inexact – floating-point inexact result
FPE_FLTDIV	fp_division – floating-point division by zero
FPE_FLTUND	fp_underflow – floating-point underflow
FPE_FLTOVF	fp_overflow – floating-point overflow
FPE_FLTINV	fp_invalid – floating-point invalid operation

Three steps are required to intercept an IEEE 754-related **SIGFPE** code with **sigfpe( )**:

1) Set up a handler with **sigfpe( )**.

2) Enable the relevant IEEE 754 trapping capability in the hardware, perhaps by using assembly-language instructions.

3) Perform a floating-point operation that generates the intended IEEE 754 exception.

sigfpe( ) never changes floating-point hardware mode bits affecting IEEE 754 trapping. No IEEE 754-related **SIGFPE** signals will be generated unless those hardware mode bits are enabled.

SIGFPE signals can be handled using **sigfpe**( ), **sigaction**(2) or **signal**(3C). In a particular program, to avoid confusion, use only one of these interfaces to handle **SIGFPE** signals.

EXAMPLES

A user-specified signal handler might look like this:

```
#include <floatingpoint.h>
#include <siginfo.h>
#include <ucontext.h>
/*
* The sample_handler prints out a message then commits suicide.
*/
void
sample_handler(int sig, siginfo_t *sip, ucontext_t *uap) {
        char *label;

        switch (sip−>si_code) {
        case FPE_FLTINV: label = "invalid operand"; break;
        case FPE_FLTRES: label = "inexact"; break;
        case FPE_FLTDIV: label = "division-by-zero"; break;
        case FPE_FLTUND: label = "underflow"; break;
        case FPE_FLTOVF: label = "overflow"; break;
        default: label = "???"; break;
        }

        fprintf(stderr, "FP exception %s (0x%x) occurred at address %p.\n",
                        label, sip−>si_code, (void *) sip−>si_addr);
        abort();
}
```

and it might be set up like this:

```
#include <floatingpoint.h>
#include <siginfo.h>
#include <ucontext.h>
extern void sample_handler(int, siginfo_t *, ucontext_t *);
main(void) {
        sigfpe_handler_type hdl, old_handler1, old_handler2;
/*
 * save current fp_overflow and fp_invalid handlers; set the new
 * fp_overflow handler to sample_handler( ) and set the new
 * fp_invalid handler to SIGFPE_ABORT (abort on invalid)
 */
        hdl = (sigfpe_handler_type) sample_handler;
        old_handler1 = sigfpe(FPE_FLTOVF, hdl);
        old_handler2 = sigfpe(FPE_FLTINV, SIGFPE_ABORT);
        . . .
/*
 * restore old fp_overflow and fp_invalid handlers
 */
```

```
                sigfpe(FPE_FLTOVF, old_handler1);
                sigfpe(FPE_FLTINV, old_handler2);
        }
```

FILES

/usr/include/floatingpoint.h
/usr/include/siginfo.h

SEE ALSO

sigaction(2), **abort**(3C), **signal**(3C), **floatingpoint**(5)

DIAGNOSTICS

sigfpe( ) returns BADSIG if *code* is not zero or a defined **SIGFPE** code.