

**NAME**

matherr - SVID math library exception handling

**SYNOPSIS**

```
#define _SVID_SOURCE /* See feature\_test\_macros\(7\)
*/
#include <math.h>

int matherr(struct exception *exc);

extern _LIB_VERSION_TYPE _LIB_VERSION;
```

Link with *-lm*.

**DESCRIPTION**

The System V Interface Definition (SVID) specifies that various math functions should invoke a function called **matherr()** if a math exception is detected. This function is called before the math function returns; after **matherr()** returns, the system then returns to the math function, which in turn returns to the caller.

The **matherr()** mechanism is supported by glibc, but is now obsolete: new applications should use the techniques described in [math\\_error\(7\)](#) and [fenv\(3\)](#). This page documents the glibc **matherr()** mechanism as an aid for maintaining and porting older applications.

To employ **matherr()**, the programmer must define the **\_SVID\_SOURCE** feature test macro (before including *any* header files), and assign the value **\_SVID\_** to the external variable **\_LIB\_VERSION**.

The system provides a default version of **matherr()**. This version does nothing, and returns zero (see below for the significance of this). The default **matherr()** can be overridden by a programmer-defined version, which will be invoked when an exception occurs. The function is invoked with one argument, a pointer to an *exception* structure, defined as follows:

```
struct exception {
    int type; /* Exception type */
    char *name; /* Name of function causing exception */
    double arg1; /* 1st argument to function */
    double arg2; /* 2nd argument to function */
    double retval; /* Function return value */
}
```

The *type* field has one of the following values:

**DOMAIN** A domain error occurred (the function argument was outside the range for which the function is defined). The return value depends on the function; *errno* is set to **EDOM**.

**SING** A pole error occurred (the function result is an infinity). The return value in most cases is **HUGE** (the largest single precision floating-point number), appropriately signed. In most cases, *errno* is set to **EDOM**.

**OVERFLOW**

An overflow occurred. In most cases, the value **HUGE** is returned, and *errno* is set to **ERANGE**.

**UNDERFLOW**

An underflow occurred. 0.0 is returned, and *errno* is set to **ERANGE**.

**TLOSS** Total loss of significance. 0.0 is returned, and *errno* is set to **ERANGE**.

**PLOSS** Partial loss of significance. This value is unused on glibc (and many other systems).

The *arg1* and *arg2* fields are the arguments supplied to the function (*arg2* is undefined for functions that take only one argument).

The *retval* field specifies the return value that the math function will return to its caller. The programmer-defined **matherr()** can modify this field to change the return value of the math function.

If the **matherr()** function returns zero, then the system sets *errno* as described above, and may print an error message on standard error (see below).

If the **matherr()** function returns a nonzero value, then the system does not set *errno*, and doesn't print an error message.

### Math functions that employ matherr()

The table below lists the functions and circumstances in which **matherr()** is called. The Type column indicates the value assigned to *exc->type* when calling **matherr()**. The Result column is the default return value assigned to *exc->retval*.

The Msg? and errno columns describe the default behavior if **matherr()** returns zero. If the Msg? column contains y, then the system prints an error message on standard error.

The table uses the following notations and abbreviations:

x first argument to function  
y second argument to function  
fin finite value for argument  
neg negative value for argument  
int integral value for argument  
o/f result overflowed  
u/f result underflowed  
|x| absolute value of x  
X\_TLOSS is a constant defined in *<math.h>*

Function	Type	Result	Msg?	errno
acos( x >1)	DOMAIN	HUGE	y	EDOM
asin( x >1)	DOMAIN	HUGE	y	EDOM
atan2(0,0)	DOMAIN	HUGE	y	EDOM
acosh(x<1)	DOMAIN	NAN	y	EDOM
atanh( x >1)	DOMAIN	NAN	y	EDOM
atanh( x =1)	SING	(x>0.0)? HUGE_VAL : -HUGE_VAL	y	EDOM
cosh(fin) o/f	OVERFLOW	HUGE	n	ERANGE
sinh(fin) o/f	OVERFLOW	(x>0.0) ? HUGE : -HUGE	n	ERANGE
sqrt(x<0)	DOMAIN	0.0	y	EDOM
hypot(fin,fin) o/f	OVERFLOW	HUGE	n	ERANGE
exp(fin) o/f	OVERFLOW	HUGE	n	ERANGE
exp(fin) u/f	UNDERFLOW	0.0	n	ERANGE
exp2(fin) o/f	OVERFLOW	HUGE	n	ERANGE
exp2(fin) u/f	UNDERFLOW	0.0	n	ERANGE
exp10(fin) o/f	OVERFLOW	HUGE	n	ERANGE
exp10(fin) u/f	UNDERFLOW	0.0	n	ERANGE
j0( x >X_TLOSS)	TLOSS	0.0	y	ERANGE
j1( x >X_TLOSS)	TLOSS	0.0	y	ERANGE
jn( x >X_TLOSS)	TLOSS	0.0	y	ERANGE
y0(x>X_TLOSS)	TLOSS	0.0	y	ERANGE
y1(x>X_TLOSS)	TLOSS	0.0	y	ERANGE
yn(x>X_TLOSS)	TLOSS	0.0	y	ERANGE
y0(0)	DOMAIN	-HUGE	y	EDOM

<code>y0(x&lt;0)</code>	DOMAIN	-HUGE	y	EDOM
<code>y1(0)</code>	DOMAIN	-HUGE	y	EDOM
<code>y1(x&lt;0)</code>	DOMAIN	-HUGE	y	EDOM
<code>yn(n,0)</code>	DOMAIN	-HUGE	y	EDOM
<code>yn(x&lt;0)</code>	DOMAIN	-HUGE	y	EDOM
<code>lgamma(fin) o/f</code>	OVERFLOW	HUGE	n	ERANGE
<code>lgamma(-int) or lgamma(0)</code>	SING	HUGE	y	EDOM
<code>tgamma(fin) o/f</code>	OVERFLOW	HUGE_VAL	n	ERANGE
<code>tgamma(-int)</code>	SING	NAN	y	EDOM
<code>tgamma(0)</code>	SING	copysign( HUGE_VAL,x)	y	ERANGE
<code>log(0)</code>	SING	-HUGE	y	EDOM
<code>log(x&lt;0)</code>	DOMAIN	-HUGE	y	EDOM
<code>log2(0)</code>	SING	-HUGE	n	EDOM
<code>log2(x&lt;0)</code>	DOMAIN	-HUGE	n	EDOM
<code>log10(0)</code>	SING	-HUGE	y	EDOM
<code>log10(x&lt;0)</code>	DOMAIN	-HUGE	y	EDOM
<code>pow(0.0,0.0)</code>	DOMAIN	0.0	y	EDOM
<code>pow(x,y) o/f</code>	OVERFLOW	HUGE	n	ERANGE
<code>pow(x,y) u/f</code>	UNDERFLOW	0.0	n	ERANGE
<code>pow(NaN,0.0)</code>	DOMAIN	x	n	EDOM
<code>0**neg</code>	DOMAIN	0.0	y	EDOM
<code>neg**non-int</code>	DOMAIN	0.0	y	EDOM
<code>scalb() o/f</code>	OVERFLOW	(x>0.0) ? HUGE_VAL : -HUGE_VAL	n	ERANGE
<code>scalb() u/f</code>	UNDERFLOW	copysign( 0.0,x)	n	ERANGE
<code>fmod(x,0)</code>	DOMAIN	x	y	EDOM
<code>remainder(x,0)</code>	DOMAIN	NAN	y	EDOM

## ATTRIBUTES

### Multithreading (see `pthread(7)`)

The `matherr()` function is thread-safe.

## EXAMPLE

The example program demonstrates the use of `matherr()` when calling `log(3)`. The program takes up to three command-line arguments. The first argument is the floating-point number to be given to `log(3)`. If the optional second argument is provided, then `_LIB_VERSION` is set to `_SVID_` so that `matherr()` is called, and the integer supplied in the command-line argument is used as the return value from `matherr()`. If the optional third command-line argument is supplied, then it specifies an alternative return value that `matherr()` should assign as the return value of the math function.

The following example run, where `log(3)` is given an argument of 0.0, does not use `matherr()`:

```
$ ./a.out 0.0
errno: Numerical result out of range
x=-inf
```

In the following run, `matherr()` is called, and returns 0:

```
$ ./a.out 0.0 0
matherr SING exception in log() function
args: 0.000000, 0.000000
retval: -340282346638528859811704183484516925440.000000
log: SING error
```

```

errno: Numerical argument out of domain
x=-340282346638528859811704183484516925440.000000

```

The message log: SING error was printed by the C library.

In the following run, `matherr()` is called, and returns a nonzero value:

```

$ ./a.out 0.0 1
matherr SING exception in log() function
args: 0.000000, 0.000000
retval: -340282346638528859811704183484516925440.000000
x=-340282346638528859811704183484516925440.000000

```

In this case, the C library did not print a message, and `errno` was not set.

In the following run, `matherr()` is called, changes the return value of the math function, and returns a nonzero value:

```

$ ./a.out 0.0 1 12345.0
matherr SING exception in log() function
args: 0.000000, 0.000000
retval: -340282346638528859811704183484516925440.000000
x=12345.000000

```

### Program source

```

#define _SVID_SOURCE
#include <errno.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

static int matherr_ret = 0; /* Value that matherr()
should return */
static int change_retval = 0; /* Should matherr() change
functions return value? */
static double new_retval; /* New function return value */

int
matherr(struct exception *exc)
{
    fprintf(stderr, matherr %s exception in %s() functionn,
(exc->type == DOMAIN) ? DOMAIN :
(exc->type == OVERFLOW) ? OVERFLOW :
(exc->type == UNDERFLOW) ? UNDERFLOW :
(exc->type == SING) ? SING :
(exc->type == TLOSS) ? TLOSS :
(exc->type == PLOSS) ? PLOSS : ???,
exc->name);
    fprintf(stderr, args: %f, %fn,
exc->arg1, exc->arg2);
    fprintf(stderr, retval: %fn, exc->retval);

    if (change_retval)
        exc->retval = new_retval;

    return matherr_ret;
}

int
main(int argc, char *argv[])

```

```
{
double x;

if (argc < 2) {
fprintf(stderr, Usage: %s <argval>
 [<matherr-ret> [<new-func-retval>]]n, argv[0]);
exit(EXIT_FAILURE);
}

if (argc > 2) {
_LIB_VERSION = _SVID_;
matherr_ret = atoi(argv[2]);
}

if (argc > 3) {
change_retval = 1;
new_retval = atof(argv[3]);
}

x = log(atof(argv[1]));
if (errno != 0)
perror(errno);

printf(x=%fn, x);
exit(EXIT_SUCCESS);
}
```

**SEE ALSO**

[fenv\(3\)](#), [math\\_error\(7\)](#), [standards\(7\)](#)

**COLOPHON**

This page is part of release 3.74 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at <http://www.kernel.org/doc/man-pages/>.