Rocky Enterprise Linux 9.2 Manual Pages on command 'matherr.3'

$ man matherr.3

MATHERR(3) Linux Programmer's Manual MATHERR(3)

NAME

matherr - SVID math library exception handling

SYNOPSIS

#include <math.h>

int matherr(struct exception *exc);

extern _LIB_VERSION_TYPE _LIB_VERSION;

Link with -lm.

DESCRIPTION

Note: the mechanism described in this page is no longer supported by glibc. Before glibc 2.27, it had been marked as obsolete. Since glibc 2.27, the mechanism has been removed altogether. New applications should use the techniques described in math_error(7) and fenv(3). This page documents the matherr() mechanism as an aid for maintaining and porting older applications.

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.

To employ matherr(), the programmer must define the _SVID_SOURCE fea? ture 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 ver? sion, 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 re? turned, 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 math? err() 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 math? err() returns zero. If the "Msg?" columns contains "y", then the sys? tem 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)? y EDOM

HUGE_VAL :

-HUGE_VAL

cosh(fin) o/f OVERFLOW HUGE n ERANGE

sinh(fin) o/f OVERFLOW (x>0.0) ? n ERANGE

HUGE : -HUGE

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

y0(x<0) DOMAIN -HUGE y EDOM

y1(0) DOMAIN -HUGE y EDOM

y1(x<0) DOMAIN -HUGE y EDOM

yn(n,0) DOMAIN -HUGE y EDOM

yn(x<0) DOMAIN -HUGE y EDOM

lgamma(fin) o/f OVERFLOW HUGE n ERANGE

lgamma(-int) or SING HUGE y EDOM

lgamma(0)

tgamma(fin) o/f OVERFLOW HUGE_VAL n ERANGE

tgamma(-int) SING NAN y EDOM

tgamma(0) SING copysign(y ERANGE
HUGE_VAL,x)

log(0) SING -HUGE y EDOM

log(x<0) DOMAIN -HUGE y EDOM

log2(0) SING -HUGE n EDOM

log2(x<0) DOMAIN -HUGE n EDOM

log10(0) SING -HUGE y EDOM

log10(x<0) DOMAIN -HUGE y EDOM

pow(0.0,0.0) DOMAIN 0.0 y EDOM

pow(x,y) o/f OVERFLOW HUGE n ERANGE

pow(x,y) u/f UNDERFLOW 0.0 n ERANGE

pow(NaN,0.0) DOMAIN x n EDOM

0**neg DOMAIN 0.0 y EDOM

neg**non-int DOMAIN 0.0 y EDOM

scalb() o/f OVERFLOW (x>0.0) ? n ERANGE
HUGE_VAL :
-HUGE_VAL

scalb() u/f UNDERFLOW copysign(n ERANGE
0.0,x)

fmod(x,0) DOMAIN x y EDOM

remainder(x,0) DOMAIN NAN y EDOM

ATTRIBUTES

For an explanation of the terms used in this section, see at?
tributes(7).

??????????????????????????????????????

?Interface ? Attribute ? Value ?

??????????????????????????????????????

?matherr() ? Thread safety ? MT-Safe ?

??????????????????????????????????????

EXAMPLES

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
                                       function's return value? */
    static double new_retval;       /* New function return value */
    int
    matherr(struct exception *exc)
    {
        fprintf(stderr, "matherr %s exception in %s() function\n",
                (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, %f\n",
                 exc->arg1, exc->arg2);
        fprintf(stderr, "        retval: %f\n", 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=%f\n", x);
    exit(EXIT_SUCCESS);
}
```

SEE ALSO

fenv(3), math_error(7), standards(7)

COLOPHON

This page is part of release 5.10 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 https://www.kernel.org/doc/man-pages/.