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

$ man muntrace.3

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

NAME

mtrace, muntrace - malloc tracing

SYNOPSIS

#include <mcheck.h>

void mtrace(void);

void muntrace(void);

DESCRIPTION

The mtrace() function installs hook functions for the memory-allocation functions (malloc(3), realloc(3) memalign(3), free(3)). These hook functions record tracing information about memory allocation and de? allocation. The tracing information can be used to discover memory leaks and attempts to free nonallocated memory in a program.

The muntrace() function disables the hook functions installed by mtrace(), so that tracing information is no longer recorded for the memory-allocation functions. If no hook functions were successfully installed by mtrace(), muntrace() does nothing.

When mtrace() is called, it checks the value of the environment vari?

able MALLOC_TRACE, which should contain the pathname of a file in which the tracing information is to be recorded. If the pathname is success? fully opened, it is truncated to zero length.

If MALLOC_TRACE is not set, or the pathname it specifies is invalid or not writable, then no hook functions are installed, and mtrace() has no effect. In set-user-ID and set-group-ID programs, MALLOC_TRACE is ig? nored, and mtrace() has no effect.

ATTRIBUTES

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

Interface	Attribute	Value
mtrace(), muntrace()	Thread safety	MT-Unsafe

CONFORMING TO

These functions are GNU extensions.

NOTES

In normal usage, mtrace() is called once at the start of execution of a program, and muntrace() is never called.

The tracing output produced after a call to mtrace() is textual, but not designed to be human readable. The GNU C library provides a Perl script, mtrace(1), that interprets the trace log and produces human-readable output. For best results, the traced program should be com? piled with debugging enabled, so that line-number information is recorded in the executable.

The tracing performed by mtrace() incurs a performance penalty (if MAL? LOC_TRACE points to a valid, writable pathname).

BUGS

The line-number information produced by mtrace(1) is not always pre? cise: the line number references may refer to the previous or following (nonblank) line of the source code.

EXAMPLES

The shell session below demonstrates the use of the mtrace() function and the mtrace(1) command in a program that has memory leaks at two different locations. The demonstration uses the following program:

```
$ cat t_mtrace.c
#include <mcheck.h>
#include <stdlib.h>
#include <stdio.h>
int
main(int argc, char *argv[])
{
    mtrace();
    for (int j = 0; j < 2; j++)
        malloc(100);            /* Never freed--a memory leak */
    calloc(16, 16);             /* Never freed--a memory leak */
    exit(EXIT_SUCCESS);
}
```

When we run the program as follows, we see that mtrace() diagnosed memory leaks at two different locations in the program:

```
$ cc -g t_mtrace.c -o t_mtrace
$ export MALLOC_TRACE=/tmp/t
$ ./t_mtrace
$ mtrace ./t_mtrace $MALLOC_TRACE
Memory not freed:
-----------------
   Address     Size     Caller
0x084c9378     0x64  at /home/cecilia/t_mtrace.c:12
0x084c93e0     0x64  at /home/cecilia/t_mtrace.c:12
0x084c9448    0x100  at /home/cecilia/t_mtrace.c:16
```

The first two messages about unfreed memory correspond to the two malloc(3) calls inside the for loop. The final message corresponds to the call to calloc(3) (which in turn calls malloc(3)).

SEE ALSO

mtrace(1), malloc(3), malloc_hook(3), mcheck(3)

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/.