NAME

mallopt − set memory allocation parameters

SYNOPSIS

#include <malloc.h>

int mallopt(int *param*, **int** *value*);

DESCRIPTION

The **mallopt**() function adjusts parameters that control the behavior of the memory-allocation functions (see malloc(3)). The *param* argument specifies the parameter to be modified, and *value* specifies the new value for that parameter.

The following values can be specified for *param*:

M_ARENA_MAX

If this parameter has a nonzero value, it defines a hard limit on the maximum number of arenas that can be created. An arena represents a pool of memory that can be used by malloc(3) (and similar) calls to service allocation requests. Arenas are thread safe and therefore may have multiple concurrent memory requests. The trade-off is between the number of threads and the number of arenas. The more arenas you have, the lower the per-thread contention, but the higher the memory usage.

The default value of this parameter is 0, meaning that the limit on the number of arenas is determined according to the setting of **M_ARENA_TEST**.

This parameter has been available since glibc 2.10 via **−−enable−experimental−malloc**, and since glibc 2.15 by default. In some versions of the allocator there was no limit on the number of created arenas (e.g., CentOS 5, RHEL 5).

When employing newer glibc versions, applications may in some cases exhibit high contention when accessing arenas. In these cases, it may be beneficial to increase **M_ARENA_MAX** to match the number of threads. This is similar in behavior to strategies taken by tcmalloc and jemalloc (e.g., per-thread allocation pools).

M_ARENA_TEST

This parameter specifies a value, in number of arenas created, at which point the system configuration will be examined to determine a hard limit on the number of created arenas. (See **M_ARENA_MAX** for the definition of an arena.)

The computation of the arena hard limit is implementation-defined and is usually calculated as a multiple of the number of available CPUs. Once the hard limit is computed, the result is final and constrains the total number of arenas.

The default value for the **M_ARENA_TEST** parameter is 2 on systems where *sizeof(long)* is 4; otherwise the default value is 8.

This parameter has been available since glibc 2.10 via **−−enable−experimental−malloc**, and since glibc 2.15 by default.

The value of **M_ARENA_TEST** is not used when **M_ARENA_MAX** has a nonzero value.

M_CHECK_ACTION

Setting this parameter controls how glibc responds when various kinds of programming errors are detected (e.g., freeing the same pointer twice). The 3 least significant bits (2, 1, and 0) of the value assigned to this parameter determine the glibc behavior, as follows:

Bit 0 If this bit is set, then print a one-line message on *stderr* that provides details about the error. The message starts with the string "*** glibc detected ***", followed by the program name, the name of the memory-allocation function in which the error was detected, a brief description of the error, and the memory address where the error was detected.

Bit 1 If this bit is set, then, after printing any error message specified by bit 0, the program is terminated by calling abort(3). In glibc versions since 2.4, if bit 0 is also set, then, between printing the error message and aborting, the program also prints a stack trace in the manner of backtrace(3), and prints the process's memory mapping in the style of */proc/[pid]/maps* (see proc(5)).

Bit 2 (since glibc 2.4)
This bit has an effect only if bit 0 is also set. If this bit is set, then the one-line message describing the error is simplified to contain just the name of the function where the error was detected and the brief description of the error.

The remaining bits in *value* are ignored.

Combining the above details, the following numeric values are meaningful for **M_CHECK_ACTION**:

0 Ignore error conditions; continue execution (with undefined results).

1 Print a detailed error message and continue execution.

2 Abort the program.

3 Print detailed error message, stack trace, and memory mappings, and abort the program.

5 Print a simple error message and continue execution.

7 Print simple error message, stack trace, and memory mappings, and abort the program.

Since glibc 2.3.4, the default value for the **M_CHECK_ACTION** parameter is 3. In glibc version 2.3.3 and earlier, the default value is 1.

Using a nonzero **M_CHECK_ACTION** value can be useful because otherwise a crash may happen much later, and the true cause of the problem is then very hard to track down.

M_MMAP_MAX
This parameter specifies the maximum number of allocation requests that may be simultaneously serviced using mmap(2). This parameter exists because some systems have a limited number of internal tables for use by mmap(2), and using more than a few of them may degrade performance.

The default value is 65,536, a value which has no special significance and which serves only as a safeguard. Setting this parameter to 0 disables the use of mmap(2) for servicing large allocation requests.

M_MMAP_THRESHOLD
For allocations greater than or equal to the limit specified (in bytes) by **M_MMAP_THRESHOLD** that can't be satisfied from the free list, the memory-allocation functions employ mmap(2) instead of increasing the program break using sbrk(2).

Allocating memory using mmap(2) has the significant advantage that the allocated memory blocks can always be independently released back to the system. (By contrast, the heap can be trimmed only if memory is freed at the top end.) On the other hand, there are some disadvantages to the use of mmap(2): deallocated space is not placed on the free list for reuse by later allocations; memory may be wasted because mmap(2) allocations must be page-aligned; and the kernel must perform the expensive task of zeroing out memory allocated via mmap(2). Balancing these factors leads to a default setting of 128*1024 for the **M_MMAP_THRESHOLD** parameter.

The lower limit for this parameter is 0. The upper limit is **DEFAULT_MMAP_THRESHOLD_MAX**: 512*1024 on 32-bit systems or *4*1024*1024*sizeof(long)* on 64-bit systems.

Note: Nowadays, glibc uses a dynamic mmap threshold by default. The initial value of the threshold is 128*1024, but when blocks larger than the current threshold and less than or equal to **DEFAULT_MMAP_THRESHOLD_MAX** are freed, the threshold is adjusted upward to the size of the freed block. When dynamic mmap thresholding is in effect, the threshold for trimming the heap is also dynamically adjusted to be twice the dynamic mmap threshold. Dynamic adjustment of the mmap threshold is disabled if any of the **M_TRIM_THRESHOLD**, **M_TOP_PAD**,

M_MMAP_THRESHOLD, or **M_MMAP_MAX** parameters is set.

M_MXFAST (since glibc 2.3)
Set the upper limit for memory allocation requests that are satisfied using "fastbins". (The measurement unit for this parameter is bytes.) Fastbins are storage areas that hold deallocated blocks of memory of the same size without merging adjacent free blocks. Subsequent reallocation of blocks of the same size can be handled very quickly by allocating from the fastbin, although memory fragmentation and the overall memory footprint of the program can increase.

The default value for this parameter is *64*sizeof(size_t)/4* (i.e., 64 on 32-bit architectures). The range for this parameter is 0 to *80*sizeof(size_t)/4*. Setting **M_MXFAST** to 0 disables the use of fastbins.

M_PERTURB (since glibc 2.4)
If this parameter is set to a nonzero value, then bytes of allocated memory (other than allocations via calloc(3)) are initialized to the complement of the value in the least significant byte of *value*, and when allocated memory is released using free(3), the freed bytes are set to the least significant byte of *value*. This can be useful for detecting errors where programs incorrectly rely on allocated memory being initialized to zero, or reuse values in memory that has already been freed.

The default value for this parameter is 0.

M_TOP_PAD
This parameter defines the amount of padding to employ when calling sbrk(2) to modify the program break. (The measurement unit for this parameter is bytes.) This parameter has an effect in the following circumstances:

* When the program break is increased, then **M_TOP_PAD** bytes are added to the sbrk(2) request.
* When the heap is trimmed as a consequence of calling free(3) (see the discussion of **M_TRIM_THRESHOLD**) this much free space is preserved at the top of the heap.

In either case, the amount of padding is always rounded to a system page boundary.

Modifying **M_TOP_PAD** is a trade-off between increasing the number of system calls (when the parameter is set low) and wasting unused memory at the top of the heap (when the parameter is set high).

The default value for this parameter is 128*1024.

M_TRIM_THRESHOLD
When the amount of contiguous free memory at the top of the heap grows sufficiently large, free(3) employs sbrk(2) to release this memory back to the system. (This can be useful in programs that continue to execute for a long period after freeing a significant amount of memory.) The **M_TRIM_THRESHOLD** parameter specifies the minimum size (in bytes) that this block of memory must reach before sbrk(2) is used to trim the heap.

The default value for this parameter is 128*1024. Setting **M_TRIM_THRESHOLD** to −1 disables trimming completely.

Modifying **M_TRIM_THRESHOLD** is a trade-off between increasing the number of system calls (when the parameter is set low) and wasting unused memory at the top of the heap (when the parameter is set high).

Environment variables

A number of environment variables can be defined to modify some of the same parameters as are controlled by **mallopt**(). Using these variables has the advantage that the source code of the program need not be changed. To be effective, these variables must be defined before the first call to a memory-allocation function. (If the same parameters are adjusted via **mallopt**(), then the **mallopt**() settings take precedence.) For security reasons, these variables are ignored in set-user-ID and set-group-ID programs.

The environment variables are as follows (note the trailing underscore at the end of the name of some

variables):

MALLOC_ARENA_MAX
Controls the same parameter as **mallopt**() **M_ARENA_MAX**.

MALLOC_ARENA_TEST
Controls the same parameter as **mallopt**() **M_ARENA_TEST**.

MALLOC_CHECK_
This environment variable controls the same parameter as **mallopt**() **M_CHECK_ACTION**. If this variable is set to a nonzero value, then a special implementation of the memory-allocation functions is used. (This is accomplished using the malloc_hook(3) feature.) This implementation performs additional error checking, but is slower than the standard set of memory-allocation functions. (This implementation does not detect all possible errors; memory leaks can still occur.)

The value assigned to this environment variable should be a single digit, whose meaning is as described for **M_CHECK_ACTION**. Any characters beyond the initial digit are ignored.

For security reasons, the effect of **MALLOC_CHECK_** is disabled by default for set-user-ID and set-group-ID programs. However, if the file */etc/suid−debug* exists (the content of the file is irrelevant), then **MALLOC_CHECK_** also has an effect for set-user-ID and set-group-ID programs.

MALLOC_MMAP_MAX_
Controls the same parameter as **mallopt**() **M_MMAP_MAX**.

MALLOC_MMAP_THRESHOLD_
Controls the same parameter as **mallopt**() **M_MMAP_THRESHOLD**.

MALLOC_PERTURB_
Controls the same parameter as **mallopt**() **M_PERTURB**.

MALLOC_TRIM_THRESHOLD_
Controls the same parameter as **mallopt**() **M_TRIM_THRESHOLD**.

MALLOC_TOP_PAD_
Controls the same parameter as **mallopt**() **M_TOP_PAD**.

RETURN VALUE

On success, **mallopt**() returns 1. On error, it returns 0.

ERRORS

On error, *errno* is *not* set.

CONFORMING TO

This function is not specified by POSIX or the C standards. A similar function exists on many System V derivatives, but the range of values for *param* varies across systems. The SVID defined options **M_MXFAST**, **M_NLBLKS**, **M_GRAIN**, and **M_KEEP**, but only the first of these is implemented in glibc.

BUGS

Specifying an invalid value for *param* does not generate an error.

A calculation error within the glibc implementation means that a call of the form:

```
mallopt(M_MXFAST, n)
```

does not result in fastbins being employed for all allocations of size up to *n*. To ensure desired results, *n* should be rounded up to the next multiple greater than or equal to *(2k+1)*sizeof(size_t)*, where *k* is an integer.

If **mallopt**() is used to set **M_PERTURB**, then, as expected, the bytes of allocated memory are initialized to the complement of the byte in *value*, and when that memory is freed, the bytes of the region are initialized to the byte specified in *value*. However, there is an off-by-*sizeof(size_t)* error in the implementation: instead of initializing precisely the block of memory being freed by the call *free(p)*, the block starting at *p+sizeof(size_t)* is initialized.

EXAMPLE

The program below demonstrates the use of **M_CHECK_ACTION**. If the program is supplied with an (integer) command-line argument, then that argument is used to set the **M_CHECK_ACTION** parameter. The program then allocates a block of memory, and frees it twice (an error).

The following shell session shows what happens when we run this program under glibc, with the default value for **M_CHECK_ACTION**:

```
$ ./a.out
main(): returned from first free() call
*** glibc detected *** ./a.out: double free or corruption (top): 0x09d30008 ***
======= Backtrace: =========
/lib/libc.so.6(+0x6c501)[0x523501]
/lib/libc.so.6(+0x6dd70)[0x524d70]
/lib/libc.so.6(cfree+0x6d)[0x527e5d]
./a.out[0x80485db]
/lib/libc.so.6(__libc_start_main+0xe7)[0x4cdce7]
./a.out[0x8048471]
======= Memory map: ========
001e4000-001fe000 r-xp 00000000 08:06 1083555    /lib/libgcc_s.so.1
001fe000-001ff000 r--p 00019000 08:06 1083555    /lib/libgcc_s.so.1
[some lines omitted]
b7814000-b7817000 rw-p 00000000 00:00 0
bff53000-bff74000 rw-p 00000000 00:00 0          [stack]
Aborted (core dumped)
```

The following runs show the results when employing other values for **M_CHECK_ACTION**:

```
$ ./a.out 1             # Diagnose error and continue
main(): returned from first free() call
*** glibc detected *** ./a.out: double free or corruption (top): 0x09cbe008 ***
main(): returned from second free() call
$ ./a.out 2             # Abort without error message
main(): returned from first free() call
Aborted (core dumped)
$ ./a.out 0             # Ignore error and continue
main(): returned from first free() call
main(): returned from second free() call
```

The next run shows how to set the same parameter using the **MALLOC_CHECK_** environment variable:

```
$ MALLOC_CHECK_=1 ./a.out
main(): returned from first free() call
*** glibc detected *** ./a.out: free(): invalid pointer: 0x092c2008 ***
main(): returned from second free() call
```

Program source

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

int
main(int argc, char *argv[])
{
char *p;

if (argc > 1) {
if (mallopt(M_CHECK_ACTION, atoi(argv[1])) != 1) {
fprintf(stderr, "mallopt() failed");
```

```
exit(EXIT_FAILURE);
}
}

p = malloc(1000);
if (p == NULL) {
fprintf(stderr, "malloc() failed");
exit(EXIT_FAILURE);
}

free(p);
printf("main(): returned from first free() call\n");

free(p);
printf("main(): returned from second free() call\n");

exit(EXIT_SUCCESS);
}
```

SEE ALSO

mmap(2), sbrk(2), mallinfo(3), malloc(3), malloc_hook(3), malloc_info(3), malloc_stats(3), malloc_trim(3), mcheck(3), mtrace(3), posix_memalign(3)

COLOPHON

This page is part of release 4.16 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/.