NAME

mallinfo – obtain memory allocation information

SYNOPSIS

#include <malloc.h>

struct mallinfo mallinfo(void);

DESCRIPTION

The **mallinfo**() function returns a copy of a structure containing information about memory allocations performed by **malloc**(3) and related functions.

Note that not all allocations are visible to **mallinfo**(); see BUGS and consider using **malloc_info**(3) instead.

The returned structure is defined as follows:

```
struct mallinfo {
    int arena;     /* Non-mmapped space allocated (bytes) */
    int ordblks;   /* Number of free chunks */
    int smblks;    /* Number of free fastbin blocks */
    int hblks;     /* Number of mmapped regions */
    int hblkhd;    /* Space allocated in mmapped regions (bytes) */
    int usmblks;   /* Maximum total allocated space (bytes) */
    int fsmblks;   /* Space in freed fastbin blocks (bytes) */
    int uordblks;  /* Total allocated space (bytes) */
    int fordblks;  /* Total free space (bytes) */
    int keepcost;  /* Top-most, releasable space (bytes) */
};
```

The fields of the *mallinfo* structure contain the following information:

arena The total amount of memory allocated by means other than **mmap**(2) (i.e., memory allocated on the heap). This figure includes both in-use blocks and blocks on the free list.

ordblks The number of ordinary (i.e., non-fastbin) free blocks.

smblks The number of fastbin free blocks (see **mallopt**(3)).

hblks The number of blocks currently allocated using **mmap**(2). (See the discussion of **M_MMAP_THRESHOLD** in **mallopt**(3).)

hblkhd The number of bytes in blocks currently allocated using **mmap**(2).

usmblks The "highwater mark" for allocated space—that is, the maximum amount of space that was ever allocated. This field is maintained only in nonthreading environments.

fsmblks The total number of bytes in fastbin free blocks.

uordblks The total number of bytes used by in-use allocations.

fordblks The total number of bytes in free blocks.

keepcost The total amount of releasable free space at the top of the heap. This is the maximum number of bytes that could ideally (i.e., ignoring page alignment restrictions, and so on) be released by **malloc_trim**(3).

ATTRIBUTES

For an explanation of the terms used in this section, see **attributes**(7).

Interface	Attribute	Value
mallinfo()	Thread safety	MT-Unsafe init const:mallopt

mallinfo() would access some global internal objects. If modify them with non-atomically, may get inconsistent results. The identifier *mallopt* in *const:mallopt* mean that **mallopt**() would modify the global internal objects with atomics, that make sure **mallinfo**() is safe enough, others modify with non-atomically maybe not.

CONFORMING TO

This function is not specified by POSIX or the C standards. A similar function exists on many System V derivatives, and was specified in the SVID.

BUGS

Information is returned for only the main memory allocation area. Allocations in other arenas are excluded. See **malloc_stats**(3) and **malloc_info**(3) for alternatives that include information about other arenas.

The fields of the *mallinfo* structure are typed as *int*. However, because some internal bookkeeping values may be of type *long*, the reported values may wrap around zero and thus be inaccurate.

EXAMPLE

The program below employs **mallinfo**() to retrieve memory allocation statistics before and after allocating and freeing some blocks of memory. The statistics are displayed on standard output.

The first two command-line arguments specify the number and size of blocks to be allocated with **malloc**(3).

The remaining three arguments specify which of the allocated blocks should be freed with **free**(3). These three arguments are optional, and specify (in order): the step size to be used in the loop that frees blocks (the default is 1, meaning free all blocks in the range); the ordinal position of the first block to be freed (default 0, meaning the first allocated block); and a number one greater than the ordinal position of the last block to be freed (default is one greater than the maximum block number). If these three arguments are omitted, then the defaults cause all allocated blocks to be freed.

In the following example run of the program, 1000 allocations of 100 bytes are performed, and then every second allocated block is freed:

```
$ ./a.out 1000 100 2
============== Before allocating blocks ==============
Total non-mmapped bytes (arena):       0
# of free chunks (ordblks):            1
# of free fastbin blocks (smblks):     0
# of mapped regions (hblks):           0
Bytes in mapped regions (hblkhd):      0
Max. total allocated space (usmblks):  0
Free bytes held in fastbins (fsmblks): 0
Total allocated space (uordblks):      0
Total free space (fordblks):           0
Topmost releasable block (keepcost):   0

============== After allocating blocks ==============
Total non-mmapped bytes (arena):       135168
# of free chunks (ordblks):            1
# of free fastbin blocks (smblks):     0
# of mapped regions (hblks):           0
Bytes in mapped regions (hblkhd):      0
Max. total allocated space (usmblks):  0
Free bytes held in fastbins (fsmblks): 0
Total allocated space (uordblks):      104000
Total free space (fordblks):           31168
Topmost releasable block (keepcost):   31168

============== After freeing blocks ==============
Total non-mmapped bytes (arena):       135168
# of free chunks (ordblks):            501
# of free fastbin blocks (smblks):     0
# of mapped regions (hblks):           0
```

```
       Bytes in mapped regions (hblkhd):      0
       Max. total allocated space (usmblks):  0
       Free bytes held in fastbins (fsmblks): 0
       Total allocated space (uordblks):      52000
       Total free space (fordblks):           83168
       Topmost releasable block (keepcost):   31168
```

Program source

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

static void
display_mallinfo(void)
{
    struct mallinfo mi;

    mi = mallinfo();

    printf("Total non-mmapped bytes (arena):       %d\n", mi.arena);
    printf("# of free chunks (ordblks):            %d\n", mi.ordblks);
    printf("# of free fastbin blocks (smblks):     %d\n", mi.smblks);
    printf("# of mapped regions (hblks):           %d\n", mi.hblks);
    printf("Bytes in mapped regions (hblkhd):      %d\n", mi.hblkhd);
    printf("Max. total allocated space (usmblks):  %d\n", mi.usmblks);
    printf("Free bytes held in fastbins (fsmblks): %d\n", mi.fsmblks);
    printf("Total allocated space (uordblks):      %d\n", mi.uordblks);
    printf("Total free space (fordblks):           %d\n", mi.fordblks);
    printf("Topmost releasable block (keepcost):   %d\n", mi.keepcost);
}

int
main(int argc, char *argv[])
{
#define MAX_ALLOCS 2000000
    char *alloc[MAX_ALLOCS];
    int numBlocks, j, freeBegin, freeEnd, freeStep;
    size_t blockSize;

    if (argc < 3 || strcmp(argv[1], "--help") == 0) {
        fprintf(stderr, "%s num-blocks block-size [free-step "
                "[start-free [end-free]]]\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    numBlocks = atoi(argv[1]);
    blockSize = atoi(argv[2]);
    freeStep = (argc > 3) ? atoi(argv[3]) : 1;
    freeBegin = (argc > 4) ? atoi(argv[4]) : 0;
    freeEnd = (argc > 5) ? atoi(argv[5]) : numBlocks;

    printf("============== Before allocating blocks ==============\n");
    display_mallinfo();
```

```
        for (j = 0; j < numBlocks; j++) {
            if (numBlocks >= MAX_ALLOCS) {
                fprintf(stderr, "Too many allocations\n");
                exit(EXIT_FAILURE);
            }

            alloc[j] = malloc(blockSize);
            if (alloc[j] == NULL) {
                perror("malloc");
                exit(EXIT_FAILURE);
            }
        }

        printf("\n============== After allocating blocks ==============\n");
        display_mallinfo();

        for (j = freeBegin; j < freeEnd; j += freeStep)
            free(alloc[j]);

        printf("\n============== After freeing blocks ==============\n");
        display_mallinfo();

        exit(EXIT_SUCCESS);
    }
```

SEE ALSO

mmap(2), **malloc**(3), **malloc_info**(3), **malloc_stats**(3), **malloc_trim**(3), **mallopt**(3)

COLOPHON

This page is part of release 5.02 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/.