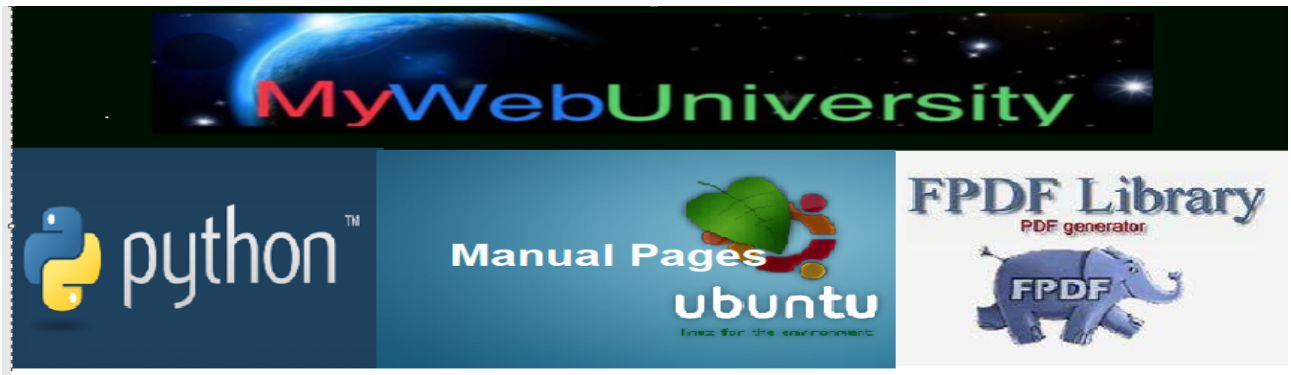

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'mallinfo.3'

$ man mallinfo.3

mallinfo(3) Library Functions Manual mallinfo(3)

NAME

mallinfo, mallinfo2 - obtain memory allocation information

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <malloc.h>

struct mallinfo mallinfo(void);

struct mallinfo2 mallinfo2(void);
```

DESCRIPTION

These functions return a copy of a structure containing information about memory allocations performed by malloc(3) and related functions. The structure returned by each function con? tains the same fields. However, the older function, mallinfo(), is deprecated since the type used for the fields is too small (see BUGS).

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

The mallinfo2 structure returned by mallinfo2() is defined as follows:

```
struct mallinfo2 {
    size_t arena;     /* Non-mmapped space allocated (bytes) */
    size_t ordblks;   /* Number of free chunks */
    size_t smblks;    /* Number of free fastbin blocks */
    size_t hblks;     /* Number of mmapped regions */
    size_t hblkhd;    /* Space allocated in mmapped regions
```

```
                    (bytes) */
        size_t usmblks;   /* See below */
        size_t fsmblks;   /* Space in freed fastbin blocks (bytes) */
        size_t uordblks;  /* Total allocated space (bytes) */
        size_t fordblks;  /* Total free space (bytes) */
        size_t keepcost;  /* Top-most, releasable space (bytes) */
    };
```

The mallinfo structure returned by the deprecated mallinfo() function is exactly the same, except that the fields are typed as int.

The structure fields 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 This field is unused, and is always 0. Historically, it was the "highwater mark" for allocated space?that is, the maximum amount of space that was ever allocated (in bytes); this field was 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(), mallinfo2()	Thread safety	MT-Unsafe init const:mallopt

??

mallinfo()/ mallinfo2() 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()/ mallinfo2() is safe enough, others modify with non-atomically maybe not.

STANDARDS

None.

HISTORY

mallinfo()

glibc 2.0. SVID.

mallinfo2()

glibc 2.33.

BUGS

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

The fields of the mallinfo structure that is returned by the older mallinfo() function 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.

EXAMPLES

The program below employs mallinfo2() to retrieve memory allocation statistics before and after allocating and freeing some blocks of memory. The statistics are displayed on stan? dard 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 omit? ted, 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_mallinfo2(void)
{
    struct mallinfo2 mi;
    mi = mallinfo2();
    printf("Total non-mmapped bytes (arena):       %zu\n", mi.arena);
    printf("# of free chunks (ordblks):            %zu\n", mi.ordblks);
    printf("# of free fastbin blocks (smblks):     %zu\n", mi.smblks);
    printf("# of mapped regions (hblks):           %zu\n", mi.hblks);
    printf("Bytes in mapped regions (hblkhd):      %zu\n", mi.hblkhd);
    printf("Max. total allocated space (usmblks):  %zu\n", mi.usmblks);
    printf("Free bytes held in fastbins (fsmblks): %zu\n", mi.fsmblks);
    printf("Total allocated space (uordblks):      %zu\n", mi.uordblks);
    printf("Total free space (fordblks):           %zu\n", mi.fordblks);
    printf("Topmost releasable block (keepcost):   %zu\n", mi.keepcost);
}
int
main(int argc, char *argv[])
{
#define MAX_ALLOCS 500000
    char *alloc[MAX_ALLOCS];
    size_t blockSize, numBlocks, freeBegin, freeEnd, freeStep;
    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_mallinfo2();
        for (size_t 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_mallinfo2();
        for (size_t j = freeBegin; j < freeEnd; j += freeStep)
            free(alloc[j]);
        printf("\n============== After freeing blocks ==============\n");
        display_mallinfo2();
        exit(EXIT_SUCCESS);
    }
```

SEE ALSO

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