NAME
malloc_info – export malloc state to a stream

LIBRARY
Standard C library (*libc*, *−lc*)

SYNOPSIS
#include <malloc.h>

int malloc_info(int *options***, FILE ****stream***);**

DESCRIPTION
The **malloc_info**() function exports an XML string that describes the current state of the memory-allocation implementation in the caller. The string is printed on the file stream *stream*. The exported string includes information about all arenas (see **malloc**(3)).

As currently implemented, *options* must be zero.

RETURN VALUE
On success, **malloc_info**() returns 0. On failure, it returns −1, and *errno* is set to indicate the error.

ERRORS
EINVAL
>   *options* was nonzero.

VERSIONS
malloc_info() was added in glibc 2.10.

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

Interface	Attribute	Value
malloc_info()	Thread safety	MT-Safe

STANDARDS
This function is a GNU extension.

NOTES
The memory-allocation information is provided as an XML string (rather than a C structure) because the information may change over time (according to changes in the underlying implementation). The output XML string includes a version field.

The **open_memstream**(3) function can be used to send the output of **malloc_info**() directly into a buffer in memory, rather than to a file.

The **malloc_info**() function is designed to address deficiencies in **malloc_stats**(3) and **mallinfo**(3).

EXAMPLES
The program below takes up to four command-line arguments, of which the first three are mandatory. The first argument specifies the number of threads that the program should create. All of the threads, including the main thread, allocate the number of blocks of memory specified by the second argument. The third argument controls the size of the blocks to be allocated. The main thread creates blocks of this size, the second thread created by the program allocates blocks of twice this size, the third thread allocates blocks of three times this size, and so on.

The program calls **malloc_info**() twice to display the memory-allocation state. The first call takes place before any threads are created or memory allocated. The second call is performed after all threads have allocated memory.

In the following example, the command-line arguments specify the creation of one additional thread, and both the main thread and the additional thread allocate 10000 blocks of memory. After the blocks of memory have been allocated, **malloc_info**() shows the state of two allocation arenas.

        $ **getconf GNU_LIBC_VERSION**

```
glibc 2.13
$ ./a.out 1 10000 100
============ Before allocating blocks ============
<malloc version="1">
<heap nr="0">
<sizes>
</sizes>
<total type="fast" count="0" size="0"/>
<total type="rest" count="0" size="0"/>
<system type="current" size="135168"/>
<system type="max" size="135168"/>
<aspace type="total" size="135168"/>
<aspace type="mprotect" size="135168"/>
</heap>
<total type="fast" count="0" size="0"/>
<total type="rest" count="0" size="0"/>
<system type="current" size="135168"/>
<system type="max" size="135168"/>
<aspace type="total" size="135168"/>
<aspace type="mprotect" size="135168"/>
</malloc>

============ After allocating blocks ============
<malloc version="1">
<heap nr="0">
<sizes>
</sizes>
<total type="fast" count="0" size="0"/>
<total type="rest" count="0" size="0"/>
<system type="current" size="1081344"/>
<system type="max" size="1081344"/>
<aspace type="total" size="1081344"/>
<aspace type="mprotect" size="1081344"/>
</heap>
<heap nr="1">
<sizes>
</sizes>
<total type="fast" count="0" size="0"/>
<total type="rest" count="0" size="0"/>
<system type="current" size="1032192"/>
<system type="max" size="1032192"/>
<aspace type="total" size="1032192"/>
<aspace type="mprotect" size="1032192"/>
</heap>
<total type="fast" count="0" size="0"/>
<total type="rest" count="0" size="0"/>
<system type="current" size="2113536"/>
<system type="max" size="2113536"/>
<aspace type="total" size="2113536"/>
<aspace type="mprotect" size="2113536"/>
</malloc>
```

Program source

```
#include <err.h>
#include <errno.h>
```

```
       #include <malloc.h>
       #include <pthread.h>
       #include <stdlib.h>
       #include <unistd.h>

       static size_t        blockSize;
       static size_t        numThreads;
       static unsigned int  numBlocks;

       static void *
       thread_func(void *arg)
       {
           int tn = (int) arg;

           /* The multiplier '(2 + tn)' ensures that each thread (including
              the main thread) allocates a different amount of memory. */

           for (unsigned int j = 0; j < numBlocks; j++)
               if (malloc(blockSize * (2 + tn)) == NULL)
                   err(EXIT_FAILURE, "malloc-thread");

           sleep(100);          /* Sleep until main thread terminates. */
           return NULL;
       }

       int
       main(int argc, char *argv[])
       {
           int       sleepTime;
           pthread_t  *thr;

           if (argc < 4) {
               fprintf(stderr,
                       "%s num-threads num-blocks block-size [sleep-time]\n",
                       argv[0]);
               exit(EXIT_FAILURE);
           }

           numThreads = atoi(argv[1]);
           numBlocks = atoi(argv[2]);
           blockSize = atoi(argv[3]);
           sleepTime = (argc > 4) ? atoi(argv[4]) : 0;

           thr = calloc(numThreads, sizeof(*thr));
           if (thr == NULL)
               err(EXIT_FAILURE, "calloc");

           printf("============ Before allocating blocks ===========\n");
           malloc_info(0, stdout);

           /* Create threads that allocate different amounts of memory. */

           for (size_t tn = 0; tn < numThreads; tn++) {
               errno = pthread_create(&thr[tn], NULL, thread_func,
```

```
                                        (void *) tn);
            if (errno != 0)
                err(EXIT_FAILURE, "pthread_create");

            /* If we add a sleep interval after the start-up of each
               thread, the threads likely won't contend for malloc
               mutexes, and therefore additional arenas won't be
               allocated (see malloc(3)). */

            if (sleepTime > 0)
                sleep(sleepTime);
        }

        /* The main thread also allocates some memory. */

        for (unsigned int j = 0; j < numBlocks; j++)
            if (malloc(blockSize) == NULL)
                err(EXIT_FAILURE, "malloc");

        sleep(2);             /* Give all threads a chance to
                                 complete allocations. */

        printf("\n============ After allocating blocks ============\n");
        malloc_info(0, stdout);

        exit(EXIT_SUCCESS);
    }
```

SEE ALSO
      mallinfo(3), **malloc**(3), **malloc_stats**(3), **mallopt**(3), **open_memstream**(3)