NAME

libmemstat - library interface to retrieve kernel memory allocator statistics

LIBRARY

Kernel Memory Allocator Statistics Library (libmemstat, -lmemstat)

SYNOPSIS

#include <sys/types.h>
#include <memstat.h>

General Functions

*const char **
memstat_strerror(*int error*);

Memory Type List Management Functions

*struct memory_type_list **
memstat_mtl_alloc(*void*);

*struct memory_type **
memstat_mtl_first(*struct memory_type_list *list*);

*struct memory_type **
memstat_mtl_next(*struct memory_type *mtp*);

*struct memory_type **
memstat_mtl_find(*struct memory_type_list *list, int allocator, const char *name*);

void
memstat_mtl_free(*struct memory_type_list *list*);

int
memstat_mtl_geterror(*struct memory_type_list *list*);

Allocator Query Functions

int
memstat_kvm_all(*struct memory_type_list *list, void *kvm_handle*);

int
memstat_kvm_malloc(*struct memory_type_list *list, void *kvm_handle*);

int
memstat_kvm_uma(*struct memory_type_list *list, void *kvm_handle*);

int
memstat_sysctl_all(*struct memory_type_list *list, int flags*);

int
memstat_sysctl_malloc(*struct memory_type_list *list, int flags*);

int
memstat_sysctl_uma(*struct memory_type_list *list, int flags*);

Memory Type Accessor Methods

*const char **
memstat_get_name(*const struct memory_type *mtp*);

int
memstat_get_allocator(*const struct memory_type *mtp*);

uint64_t
memstat_get_countlimit(*const struct memory_type *mtp*);

uint64_t
memstat_get_byteslimit(*const struct memory_type *mtp*);

uint64_t
memstat_get_sizemask(*const struct memory_type *mtp*);

uint64_t
memstat_get_size(*const struct memory_type *mtp*);

uint64_t
memstat_get_rsize(*const struct memory_type *mtp*);

uint64_t
memstat_get_memalloced(*const struct memory_type *mtp*);

uint64_t
memstat_get_memfreed(*const struct memory_type *mtp*);

uint64_t
memstat_get_numallocs(*const struct memory_type *mtp*);

uint64_t
memstat_get_numfrees(*const struct memory_type *mtp*);

uint64_t
memstat_get_bytes(*const struct memory_type *mtp*);

uint64_t
memstat_get_count(*const struct memory_type *mtp*);

uint64_t
memstat_get_free(*const struct memory_type *mtp*);

uint64_t
memstat_get_failures(*const struct memory_type *mtp*);

*void **
memstat_get_caller_pointer(*const struct memory_type *mtp, int index*);

void
memstat_set_caller_pointer(*struct memory_type *mtp, int index, void *value*);

uint64_t
memstat_get_caller_uint64(*const struct memory_type *mtp, int index*);

void
memstat_set_caller_uint64(*struct memory_type *mtp, int index, uint64_t value*);

uint64_t
memstat_get_zonefree(*const struct memory_type *mtp*);

uint64_t
memstat_get_kegfree(*const struct memory_type *mtp*);

uint64_t
memstat_get_percpu_memalloced(*const struct memory_type *mtp, int cpu*);

uint64_t

memstat_get_percpu_memfreed(*const struct memory_type *mtp, int cpu*);

uint64_t
memstat_get_percpu_numallocs(*const struct memory_type *mtp, int cpu*);

uint64_t
memstat_get_percpu_numfrees(*const struct memory_type *mtp, int cpu*);

uint64_t
memstat_get_percpu_sizemask(*const struct memory_type *mtp, int cpu*);

*void **
memstat_get_percpu_caller_pointer(*const struct memory_type *mtp, int cpu, int index*);

void
memstat_set_percpu_caller_pointer(*struct memory_type *mtp, int cpu, int index, void *value*);

uint64_t
memstat_get_percpu_caller_uint64(*const struct memory_type *mtp, int cpu, int index*);

void
memstat_set_percpu_caller_uint64(*struct memory_type *mtp, int cpu, int index, uint64_t value*);

uint64_t
memstat_get_percpu_free(*const struct memory_type *mtp, int cpu*);

DESCRIPTION

libmemstat provides an interface to retrieve kernel memory allocator statistics, for the purposes of debugging and system monitoring, insulating applications from implementation details of the allocators, and allowing a tool to transparently support multiple allocators. **libmemstat** supports both retrieving a single statistics snapshot, as well as incrementally updating statistics for long-term monitoring.

libmemstat describes each memory type using a *struct memory_type*, an opaque memory type accessed by the application using accessor functions in the library. **libmemstat** returns and updates chains of *struct memory_type* via a *struct memory_type_list*, which will be allocated by calling **memstat_mtl_alloc**(), and freed on completion using **memstat_mtl_free**(). Lists of memory types are populated via calls that query the kernel for statistics information; currently: **memstat_kvm_all**(), **memstat_kvm_malloc**(), **memstat_kvm_uma**(), **memstat_sysctl_all**(), **memstat_sysctl_uma**(), and **memstat_sysctl_malloc**(). Repeated calls will incrementally update the list of memory types, permitting tracking over time without recreating all list state. If an error is detected during a query call, error

condition information may be retrieved using **memstat_mtl_geterror**(), and converted to a user-readable string using **memstat_strerror**().

Freeing the list will free all memory type data in the list, and so invalidates any outstanding pointers to entries in the list. *struct memory_type* entries in the list may be iterated over using **memstat_mtl_first**() and **memstat_mtl_next**(), which respectively return the first entry in a list, and the next entry in a list. **memstat_mtl_find**(), which will return a pointer to the first entry matching the passed parameters.

A series of accessor methods is provided to access fields of the structure, including retrieving statistics and properties, as well as setting of caller owned fields. Direct application access to the data structure fields is not supported.

Library *memory_type Ss Fields*

Each *struct memory_type* holds a description of the memory type, including its name and the allocator it is managed by, as well as current statistics on use. Some statistics are directly measured, others are derived from directly measured statistics. Certain high level statistics are present across all available allocators, such as the number of allocation and free operations; other measurements, such as the quantity of free items in per-CPU caches, or administrative limit on the number of allocations, is available only for specific allocators.

Caller *memory_type Ss Fields*

struct memory_type includes fields to allow the application to store data, in the form of pointers and 64-bit integers, with memory types. For example, the application author might make use of one of the caller pointers to reference a more complex data structure tracking long-term behavior of the memory type, or a window system object that is used to render the state of the memory type. General and per-CPU storage is provided with each *struct memory_type* in the form of an array of pointers and integers. The array entries are accessed via the *index* argument to the get and set accessor methods. Possible values of *index* range between 0 and MEMSTAT_MAXCALLER.

Caller-owned fields are initialized to 0 or NULL when a new *struct memory_type* is allocated and attached to a memory type list; these fields retain their values across queries that update library-owned fields.

Allocator Types

Currently, **libmemstat** supports two kernel allocators: ALLOCATOR_UMA for uma(9), and ALLOCATOR_MALLOC for malloc(9). These values may be passed to **memstat_mtl_find**(), and will be returned by **memstat_get_allocator**(). Two additional constants in the allocator name space are defined: ALLOCATOR_UNKNOWN, which will only be returned as a result of a library error, and ALLOCATOR_ANY, which can be used to specify that returning types matching any allocator is permittable from **memstat_mtl_find**().

Access Method List

The following accessor methods are defined, of which some will be valid for a given memory type:

memstat_get_name()
: Return a pointer to the name of the memory type. Memory for the name is owned by **libmemstat** and will be valid through a call to **memstat_mtl_free**(). Note that names will be unique with respect to a single allocator, but that the same name might be used by different memory types owned by different memory allocators.

memstat_get_allocator()
: Return an integer identifier for the memory allocator that owns the memory type.

memstat_get_countlimit()
: If the memory type has an administrative limit on the number of simultaneous allocations, return it.

memstat_get_byteslimit()
: If the memory type has an administrative limit on the number of bytes of memory that may be simultaneously allocated for the memory type, return it.

memstat_get_sizemask()
: If the memory type supports variable allocation sizes, return a bitmask of sizes allocated for the memory type.

memstat_get_size()
: If the memory type supports a fixed allocation size, return that size.

memstat_get_rsize()
: If the memory type supports a fixed allocation size, return real size of an allocation. Real size can exceed requested size due to alignment constraints or implicit padding.

memstat_get_memalloced()
: Return the total number of bytes allocated for the memory type over its lifetime.

memstat_get_memfreed()
: Return the total number of bytes freed for the memory type over its lifetime.

memstat_get_numallocs()
: Return the total number of allocations for the memory type over its lifetime.

memstat_get_numfrees()
Return the total number of frees for the memory type over its lifetime.

memstat_get_bytes()
Return the current number of bytes allocated to the memory type.

memstat_get_count()
Return the current number of allocations for the memory type.

memstat_get_free()
If the memory allocator supports a cache, return the number of items in the cache.

memstat_get_failures()
If the memory allocator and type permit allocation failures, return the number of allocation failures measured.

memstat_get_caller_pointer()
Return a caller-owned pointer for the memory type.

memstat_set_caller_pointer()
Set a caller-owned pointer for the memory type.

memstat_get_caller_uint64()
Return a caller-owned integer for the memory type.

memstat_set_caller_uint64()
Set a caller-owned integer for the memory type.

memstat_get_zonefree()
If the memory allocator supports a multi-level allocation structure, return the number of cached items in the zone. These items will be in a fully constructed state available for immediate use.

memstat_get_kegfree()
If the memory allocator supports a multi-level allocation structure, return the number of cached items in the keg. These items may be in a partially constructed state, and may require further processing before they can be made available for use.

memstat_get_percpu_memalloced()
If the memory allocator supports per-CPU statistics, return the number of bytes of memory allocated for the memory type on the CPU over its lifetime.

memstat_get_percpu_memfreed()
If the memory allocator supports per-CPU statistics, return the number of bytes of memory freed from the memory type on the CPU over its lifetime.

memstat_get_percpu_numallocs()
If the memory allocator supports per-CPU statistics, return the number of allocations for the memory type on the CPU over its lifetime.

memstat_get_percpu_numfrees()
If the memory allocator supports per-CPU statistics, return the number of frees for the memory type on the CPU over its lifetime.

memstat_get_percpu_sizemask()
If the memory allocator supports variable size memory allocation and per-CPU statistics, return the size bitmask for the memory type on the CPU.

memstat_get_percpu_caller_pointer()
Return a caller-owned per-CPU pointer for the memory type.

memstat_set_percpu_caller_pointer()
Set a caller-owned per-CPU pointer for the memory type.

memstat_get_percpu_caller_uint64()
Return a caller-owned per-CPU integer for the memory type.

memstat_set_percpu_caller_uint64()
Set a caller-owned per-CPU integer for the memory type.

memstat_get_percpu_free()
If the memory allocator supports a per-CPU cache, return the number of free items in the per-CPU cache of the designated CPU.

RETURN VALUES

libmemstat functions fall into three categories: functions returning a pointer to an object, functions returning an integer return value, and functions implementing accessor methods returning data from a *struct memory_type*.

Functions returning a pointer to an object will generally return NULL on failure. **memstat_mtl_alloc**() will return an error value via *errno*, which will consist of the value ENOMEM. Functions **memstat_mtl_first**(), **memstat_mtl_next**(), and **memstat_mtl_find**() will return NULL when there is no

entry or match in the list; however, this is not considered a failure mode and no error value is available.

Functions returning an integer success value will return 0 on success, or -1 on failure. If a failure is returned, the list error access method, **memstat_mtl_geterror**(), may be used to retrieve the error state. The string representation of the error may be retrieved using **memstat_strerror**(). Possible error values are:

MEMSTAT_ERROR_UNDEFINED	Undefined error. Occurs if **memstat_mtl_geterror**() is called on a list before an error associated with the list has occurred.
MEMSTAT_ERROR_NOMEMORY	Insufficient memory. Occurs if library calls to malloc(3) fail, or if a system call to retrieve kernel statistics fails with ENOMEM.
MEMSTAT_ERROR_VERSION	Returned if the current version of **libmemstat** is unable to interpret the statistics data returned by the kernel due to an explicit version mismatch, or to differences in data structures that cannot be reconciled.
MEMSTAT_ERROR_PERMISSION	Returned if a statistics source returns *errno* values of EACCES or EPERM.
MEMSTAT_ERROR_DATAERROR	Returned if **libmemstat** is unable to interpret statistics data returned by the data source, even though there does not appear to be a version problem.
MEMSTAT_ERROR_KVM	Returned if **libmemstat** experiences an error while using kvm(3) interfaces to query statistics data. Use kvm_geterr(3) to retrieve the error.
MEMSTAT_ERROR_KVM_NOSYMBOL	Returned if **libmemstat** is unable to read a required symbol from the kernel being operated on.
MEMSTAT_ERROR_KVM_SHORTREAD	Returned if **libmemstat** attempts to read data from a live memory image or kernel core dump and insufficient data is returned.

Finally, functions returning data from a *struct memory_type* pointer are not permitted to fail, and directly return either a statistic or pointer to a string.

EXAMPLES

Create a memory type list, query the uma(9) memory allocator for available statistics, and print out the number of allocations performed by the mbuf zone.

```
struct memory_type_list *mtlp;
struct memory_type *mtp;
uint64_t mbuf_count;

mtlp = memstat_mtl_alloc();
if (mtlp == NULL)
   err(-1, "memstat_mtl_alloc");
if (memstat_sysctl_uma(mtlp, 0) < 0)
   err(-1, "memstat_sysctl_uma");
mtp = memstat_mtl_find(mtlp, ALLOCATOR_UMA, "mbuf");
if (mtp == NULL)
   errx(-1, "memstat_mtl_find: mbuf not found");
mbuf_count = memstat_get_count(mtp);
memstat_mtl_free(mtlp);

printf("mbufs: %llu\n", (unsigned long long)mbuf_count);
```

SEE ALSO

malloc(9), uma(9)

HISTORY

The **libmemstat** library appeared in FreeBSD 6.0.

AUTHORS

The kernel memory allocator changes necessary to support a general purpose monitoring library, along with the library, were written by Robert Watson <*rwatson@FreeBSD.org*>.

BUGS

There are memory allocators in the kernel, such as the VM page allocator and **sf_buf** allocator, which are not currently supported by **libmemstat**.

Once a memory type is present on a memory type list, it will not be removed even if the kernel no longer presents information on the type via its monitoring interfaces. In order to flush removed memory types, it is necessary to free the entire list and allocate a new one.