NAME

kmem_alloc, kmem_zalloc, kmem_free – allocate kernel memory

SYNOPSIS

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

void ∗**kmem_alloc(size_t** *size*, **int** *flag***);**

void ∗**kmem_zalloc(size_t** *size*, **int** *flag***);**

void kmem_free(void ∗*buf*, **size_t** *size***);**

INTERFACE LEVEL

Architecture independent level 1 (DDI/DKI).

ARGUMENTS

size Number of bytes to allocate.

flag Determines whether caller can sleep for memory. Possible flags are **KM_SLEEP** to allow sleeping until memory is available, or **KM_NOSLEEP** to return **NULL** immediately if memory is not available.

buf Pointer to allocated memory.

DESCRIPTION

kmem_alloc( ) allocates *size* bytes of kernel memory and returns a pointer to the allocated memory. The allocated memory is at least double-word aligned, so it can hold any **C** data structure. No greater alignment can be assumed. *flag* determines whether the caller can sleep for memory. **KM_SLEEP** allocations may sleep but are guaranteed to succeed. **KM_NOSLEEP** allocations are guaranteed not to sleep but may fail (return **NULL**) if no memory is currently available. The initial contents of memory allocated using **kmem_alloc**( ) are random garbage.

kmem_zalloc( ) is like **kmem_alloc**( ) but returns zero-filled memory.

kmem_free( ) frees previously allocated kernel memory. The buffer address and size must *exactly* match the original allocation. Memory cannot be returned piecemeal.

RETURN VALUES

If successful, **kmem_alloc**( ) and **kmem_zalloc**( ) return a pointer to the allocated memory. If **KM_NOSLEEP** is set and memory cannot be allocated without sleeping, **kmem_alloc**( ) and **kmem_zalloc**( ) return **NULL.**

CONTEXT

kmem_alloc( ) and **kmem_zalloc**( ) can be called from interrupt context only if the **KM_NOSLEEP** flag is set. They can be called from user context with any valid *flag*. **kmem_free**( ) can be called from user or interrupt context.

SEE ALSO

freerbuf(9F), **getrbuf**(9F)

Writing Device Drivers

WARNINGS

Memory allocated using **kmem_alloc**( ) is not paged. Available memory is therefore limited by the total physical memory on the system. It is also limited by the available kernel virtual address space, which is often the more restrictive constraint on large-memory configurations.

Excessive use of kernel memory is likely to affect overall system performance. Overcommitment of kernel memory will cause the system to hang or panic.

Misuse of the kernel memory allocator — such as writing past the end of a buffer, using a buffer after freeing it, freeing a buffer twice, or freeing a NULL or invalid pointer — will corrupt the kernel heap and may cause the system to corrupt data and/or panic.

The initial contents of memory allocated using **kmem_alloc**( ) are random garbage. This random garbage may include secure kernel data. Therefore, uninitialized kernel memory should be handled carefully. For example, never **copyout**(9F) a potentially uninitialized buffer.

NOTES

kmem_alloc(0, *flag***)** always returns **NULL**. **kmem_free(NULL, 0)** is legal.