NAME

osd, **osd_register**, **osd_deregister**, **osd_set**, **osd_reserve**, **osd_set_reserved**, **osd_free_reserved**, **osd_get**, **osd_del**, **osd_call**, **osd_exit** - Object Specific Data

SYNOPSIS

#include <sys/osd.h>

typedef void
(*osd_destructor_t)(*void *value*);

typedef int
(*osd_method_t)(*void *obj*, *void *data*);

int
osd_register(*u_int type*, *osd_destructor_t destructor*, *osd_method_t *methods*);

void
osd_deregister(*u_int type*, *u_int slot*);

int
osd_set(*u_int type*, *struct osd *osd*, *u_int slot*, *void *value*);

*void **
osd_reserve(*u_int slot*);

int
osd_set_reserved(*u_int type*, *struct osd *osd*, *u_int slot*, *void **rsv*, *void *value*);

void
osd_free_reserved(*void **rsv*);

*void **
osd_get(*u_int type*, *struct osd *osd*, *u_int slot*);

void
osd_del(*u_int type*, *struct osd *osd*, *u_int slot*);

int
osd_call(*u_int type*, *u_int method*, *void *obj*, *void *data*);

void
osd_exit(*u_int type*, *struct osd *osd*);

DESCRIPTION

The **osd** framework provides a mechanism to dynamically associate arbitrary data at run-time with any kernel data structure which has been suitably modified for use with **osd**. The one-off modification required involves embedding a *struct osd* inside the kernel data structure.

An additional benefit is that after the initial change to a structure is made, all subsequent use of **osd** with the structure involves no changes to the structure's layout. By extension, if the data structure is part of the ABI, **osd** provides a way of extending the structure in an ABI preserving manner.

The details of the embedded *struct osd* are not relevant to consumers of the **osd** framework and should not be manipulated directly.

Data associated with a structure is referenced by the **osd** framework using a type/slot identifier pair. Types are statically defined in <*sys/osd.h*> and provide a high-level grouping for slots to be registered under. Slot identifiers are dynamically assigned by the framework when a data type is registered using **osd_register**() and remains valid until a corresponding call to **osd_deregister**().

Functions

The **osd_register**() function registers a type/slot identifier pair with the **osd** framework for use with a new data type. The function may sleep and therefore cannot be called from a non-sleepable context. The *type* argument specifies which high-level type grouping from <*sys/osd.h*> the slot identifier should be allocated under. The *destructor* argument specifies an optional osd_destructor_t function pointer that will be called for objects of the type being registered which are later destroyed by the **osd_del**() function. NULL may be passed if no destructor is required. The *methods* argument specifies an optional array of osd_method_t function pointers which can be later invoked by the **osd_call**() function. NULL may be passed if no methods are required. The *methods* argument is currently only useful with the OSD_JAIL type identifier.

The **osd_deregister**() function deregisters a previously registered type/slot identifier pair. The function may sleep and therefore cannot be called from a non-sleepable context. The *type* argument specifies which high-level type grouping from <*sys/osd.h*> the slot identifier is allocated under. The *slot* argument specifies the slot identifier which is being deregistered and should be the value that was returned by **osd_register**() when the data type was registered.

The **osd_set**() function associates a data object pointer with a kernel data structure's *struct osd* member. The *type* argument specifies which high-level type grouping from <*sys/osd.h*> the slot identifier is allocated under. The *osd* argument is a pointer to the kernel data structure's *struct osd* which will have

the *value* pointer associated with it. The *slot* argument specifies the slot identifier to assign the *value* pointer to. The *value* argument points to a data object to associate with *osd*.

The **osd_set_reserved**() function does the same as **osd_set**(), but with an extra argument *rsv* that is internal-use memory previously allocated via **osd_reserve**().

The **osd_get**() function returns the data pointer associated with a kernel data structure's *struct osd* member from the specified type/slot identifier pair. The *type* argument specifies which high-level type grouping from <*sys/osd.h*> the slot identifier is allocated under. The *osd* argument is a pointer to the kernel data structure's *struct osd* to retrieve the data pointer from. The *slot* argument specifies the slot identifier to retrieve the data pointer from.

The **osd_del**() function removes the data pointer associated with a kernel data structure's *struct osd* member from the specified type/slot identifier pair. The *type* argument specifies which high-level type grouping from <*sys/osd.h*> the slot identifier is allocated under. The *osd* argument is a pointer to the kernel data structure's *struct osd* to remove the data pointer from. The *slot* argument specifies the slot identifier to remove the data pointer from. If an osd_destructor_t function pointer was specified at registration time, the destructor function will be called and passed the data pointer for the type/slot identifier pair which is being deleted.

The **osd_call**() function calls the specified osd_method_t function pointer for all currently registered slots of a given type on the specified *obj* and *data* pointers. The function may sleep and therefore cannot be called from a non-sleepable context. The *type* argument specifies which high-level type grouping from <*sys/osd.h*> to call the method for. The *method* argument specifies the index into the osd_method_t array that was passed to **osd_register**(). The *obj* and *data* arguments are passed to the method function pointer of each slot.

The **osd_exit**() function removes all data object pointers from all currently registered slots for a given type for the specified kernel data structure's *struct osd* member. The *type* argument specifies which high-level type grouping from <*sys/osd.h*> to remove data pointers from. The *osd* argument is a pointer to the kernel data structure's *struct osd* to remove all data object pointers for all currently registered slots from.

IMPLEMENTATION NOTES

osd uses a two dimensional matrix (array of arrays) as the data structure to manage the external data associated with a kernel data structure's *struct osd* member. The type identifier is used as the index into the outer array, and the slot identifier is used as the index into the inner array. To set or retrieve a data pointer for a given type/slot identifier pair, **osd_set**() and **osd_get**() perform the equivalent of array[type][slot], which is both constant time and fast.

If **osd_set**() is called on a *struct osd* for the first time, the array for storing data pointers is dynamically allocated using malloc(9) with M_NOWAIT to a size appropriate for the slot identifier being set. If a subsequent call to **osd_set**() attempts to set a slot identifier which is numerically larger than the slot used in the previous **osd_set**() call, realloc(9) is used to grow the array to the appropriate size such that the slot identifier can be used. To maximise the efficiency of any code which calls **osd_set**() sequentially on a number of different slot identifiers (e.g., during an initialisation phase) one should loop through the slot identifiers in descending order from highest to lowest. This will result in only a single malloc(9) call to create an array of the largest slot size and all subsequent calls to **osd_set**() will proceed without any realloc(9) calls.

It is possible for **osd_set**() to fail to allocate this array. To ensure that such allocation succeeds, **osd_reserve**() may be called (in a non-blocking context), and it will pre-allocate the memory via malloc(9) with M_WAITOK. Then this pre-allocated memory is passed to **osd_set_reserved**(), which will use it if necessary or otherwise discard it. The memory may also be explicitly discarded by calling **osd_free_reserved**(). As this method always allocates memory whether or not it is ultimately needed, it should be used only rarely, such as in the unlikely event that **osd_set**() fails.

The **osd** API is geared towards slot identifiers storing pointers to the same underlying data structure type for a given **osd** type identifier. This is not a requirement, and khelp(9) for example stores completely different data types in slots under the OSD_KHELP type identifier.

Locking

osd internally uses a mix of mutex(9), rmlock(9) and sx(9) locks to protect its internal data structures and state.

Responsibility for synchronising access to a kernel data structure's *struct osd* member is left to the subsystem that uses the data structure and calls the **osd** API.

osd_get() only acquires an rmlock in read mode, therefore making it safe to use in the majority of contexts within the kernel including most fast paths.

RETURN VALUES

osd_register() returns the slot identifier for the newly registered data type.

osd_set() and **osd_set_reserved**() return zero on success or ENOMEM if the specified type/slot identifier pair triggered an internal realloc(9) which failed (**osd_set_reserved**() will always succeed when *rsv* is non-NULL).

osd_get() returns the data pointer for the specified type/slot identifier pair, or NULL if the slot has not been initialised yet.

osd_reserve() returns a pointer suitable for passing to **osd_set_reserved**() or **osd_free_reserved**().

osd_call() returns zero if no method is run or the method for each slot runs successfully. If a method for a slot returns non-zero, **osd_call**() terminates prematurely and returns the method's error to the caller.

SEE ALSO

khelp(9)

HISTORY

The Object Specific Data (OSD) facility first appeared in FreeBSD 8.0.

AUTHORS

The **osd** facility was written by Pawel Jakub Dawidek <*pjd@FreeBSD.org*>.

This manual page was written by Lawrence Stewart <*lstewart@FreeBSD.org*>.