NAME

sysctl_ctx_init, **sysctl_ctx_free**, **sysctl_ctx_entry_add**, **sysctl_ctx_entry_find**, **sysctl_ctx_entry_del** - sysctl context for managing dynamically created sysctl OIDs

SYNOPSIS

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

int
sysctl_ctx_init(*struct sysctl_ctx_list *clist*);

int
sysctl_ctx_free(*struct sysctl_ctx_list *clist*);

*struct sysctl_ctx_entry **
sysctl_ctx_entry_add(*struct sysctl_ctx_list *clist, struct sysctl_oid *oidp*);

*struct sysctl_ctx_entry **
sysctl_ctx_entry_find(*struct sysctl_ctx_list *clist, struct sysctl_oid *oidp*);

int
sysctl_ctx_entry_del(*struct sysctl_ctx_list *clist, struct sysctl_oid *oidp*);

DESCRIPTION

These functions provide an interface for managing dynamically created OIDs. The sysctl context is responsible for keeping track of created OIDs, as well as their proper removal when needed. It adds a simple transactional aspect to OID removal operations; i.e., if a removal operation fails part way, it is possible to roll back the sysctl tree to its previous state.

The **sysctl_ctx_init**() function initializes a sysctl context. The *clist* argument must point to an already allocated variable. A context *must* be initialized before use. Once it is initialized, a pointer to the context can be passed as an argument to all the *SYSCTL_ADD_** macros (see sysctl_add_oid(9)), and it will be updated with entries pointing to newly created OIDS.

Internally, the context is represented as a queue(3) TAILQ linked list. The list consists of struct sysctl_ctx_entry entries:

```
struct sysctl_ctx_entry {
        struct sysctl_oid *entry;
        TAILQ_ENTRY(sysctl_ctx_entry) link;
```

```
};

TAILQ_HEAD(sysctl_ctx_list, sysctl_ctx_entry);
```

Each context entry points to one dynamic OID that it manages. Newly created OIDs are always inserted in the front of the list.

The **sysctl_ctx_free**() function removes the context and associated OIDs it manages. If the function completes successfully, all managed OIDs have been unregistered (removed from the tree) and freed, together with all their allocated memory, and the entries of the context have been freed as well.

The removal operation is performed in two steps. First, for each context entry, the function sysctl_remove_oid(9) is executed, with parameter *del* set to 0, which inhibits the freeing of resources. If there are no errors during this step, **sysctl_ctx_free**() proceeds to the next step. If the first step fails, all unregistered OIDs associated with the context are registered again.

Note: in most cases, the programmer specifies OID_AUTO as the OID number when creating an OID. However, during registration of the OID in the tree, this number is changed to the first available number greater than or equal to CTL_AUTO_START. If the first step of context deletion fails, re-registration of the OID does not change the already assigned OID number (which is different from OID_AUTO). This ensures that re-registered entries maintain their original positions in the tree.

The second step actually performs the deletion of the dynamic OIDs. sysctl_remove_oid(9) iterates through the context list, starting from beginning (i.e., the newest entries). *Important*: this time, the function not only deletes the OIDs from the tree, but also frees their memory (provided that oid_refcnt == 0), as well as the memory of all context entries.

The **sysctl_ctx_entry_add**() function allows the addition of an existing dynamic OID to a context.

The **sysctl_ctx_entry_del**() function removes an entry from the context. *Important*: in this case, only the corresponding struct sysctl_ctx_entry is freed, but the *oidp* pointer remains intact. Thereafter, the programmer is responsible for managing the resources allocated to this OID.

The **sysctl_ctx_entry_find**() function searches for a given *oidp* within a context list, either returning a pointer to the *struct sysctl_ctx_entry* found, or NULL.

EXAMPLES

The following is an example of how to create a new top-level category and how to hook up another subtree to an existing static node. This example uses contexts to keep track of the OIDs.

```
#include <sys/sysctl.h>
 ...
static struct sysctl_ctx_list clist;
static struct sysctl_oid *oidp;
static int a_int;
static const char *string = "dynamic sysctl";
 ...

sysctl_ctx_init(&clist);
oidp = SYSCTL_ADD_ROOT_NODE(&clist,
        OID_AUTO, "newtree", CTLFLAG_RW, 0, "new top level tree");
oidp = SYSCTL_ADD_INT(&clist, SYSCTL_CHILDREN(oidp),
        OID_AUTO, "newint", CTLFLAG_RW, &a_int, 0, "new int leaf");
 ...
oidp = SYSCTL_ADD_NODE(&clist, SYSCTL_STATIC_CHILDREN(_debug),
        OID_AUTO, "newtree", CTLFLAG_RW, 0, "new tree under debug");
oidp = SYSCTL_ADD_STRING(&clist, SYSCTL_CHILDREN(oidp),
        OID_AUTO, "newstring", CTLFLAG_RD, string, 0, "new string leaf");
 ...
/* Now we can free up the OIDs */
if (sysctl_ctx_free(&clist)) {
        printf("can't free this context - other OIDs depend on it");
        return (ENOTEMPTY);
} else {
        printf("Success!\n");
        return (0);
}
```

This example creates the following subtrees:

```
debug.newtree.newstring
newtree.newint
```

Note that both trees are removed, and their resources freed, through one **sysctl_ctx_free**() call, which starts by freeing the newest entries (leaves) and then proceeds to free the older entries (in this case the nodes).

SEE ALSO

queue(3), sysctl(8), sysctl(9), sysctl_add_oid(9), sysctl_remove_oid(9)

HISTORY

These functions first appeared in FreeBSD 4.2.

AUTHORS

Andrzej Bialecki <*abial@FreeBSD.org*>

BUGS

The current removal algorithm is somewhat heavy. In the worst case, all OIDs need to be unregistered, registered again, and then unregistered and deleted. However, the algorithm does guarantee transactional properties for removal operations.

All operations on contexts involve linked list traversal. For this reason, creation and removal of entries is relatively costly.