NAME

khelp, **khelp_init_osd**, **khelp_destroy_osd**, **khelp_get_id**, **khelp_get_osd**, **khelp_add_hhook**, **khelp_remove_hhook**, **KHELP_DECLARE_MOD**, **KHELP_DECLARE_MOD_UMA** - Kernel Helper Framework

SYNOPSIS

#include <sys/khelp.h>
#include <sys/module_khelp.h>

int khelp_init_osd(*uint32_t classes, struct osd *hosd*);

int khelp_destroy_osd(*struct osd *hosd*);

int32_t khelp_get_id(*char *hname*);

void * khelp_get_osd(*struct osd *hosd, int32_t id*);

int khelp_add_hhook(*struct hookinfo *hki, uint32_t flags*);

int khelp_remove_hhook(*struct hookinfo *hki*);

KHELP_DECLARE_MOD(*hname, hdata, hhooks, version*);

KHELP_DECLARE_MOD_UMA(*hname, hdata, hhooks, version, ctor, dtor*);

DESCRIPTION

khelp provides a framework for managing **khelp** modules, which indirectly use the hhook(9) KPI to register their hook functions with hook points of interest within the kernel. Khelp modules aim to provide a structured way to dynamically extend the kernel at runtime in an ABI preserving manner. Depending on the subsystem providing hook points, a **khelp** module may be able to associate per-object data for maintaining relevant state between hook calls. The hhook(9) and **khelp** frameworks are tightly integrated and anyone interested in **khelp** should also read the hhook(9) manual page thoroughly.

Information for Khelp Module Implementors

khelp modules are represented within the **khelp** framework by a *struct helper* which has the following members:

```
struct helper {
        int (*mod_init) (void);
        int (*mod_destroy) (void);
```

```
#define HELPER_NAME_MAXLEN 16
        char                    h_name[HELPER_NAME_MAXLEN];
        uma_zone_t              h_zone;
        struct hookinfo         *h_hooks;
        uint32_t        h_nhooks;
        uint32_t        h_classes;
        int32_t                 h_id;
        volatile uint32_t       h_refcount;
        uint16_t        h_flags;
        TAILQ_ENTRY(helper)     h_next;
};
```

Modules must instantiate a *struct helper*, but are only required to set the *h_classes* field, and may optionally set the *h_flags*, *mod_init* and *mod_destroy* fields where required. The framework takes care of all other fields and modules should refrain from manipulating them. Using the C99 designated initialiser feature to set fields is encouraged.

If specified, the *mod_init* function will be run by the **khelp** framework prior to completing the registration process. Returning a non-zero value from the *mod_init* function will abort the registration process and fail to load the module. If specified, the *mod_destroy* function will be run by the **khelp** framework during the deregistration process, after the module has been deregistered by the **khelp** framework. The return value is currently ignored. Valid **khelp** classes are defined in <*sys/khelp.h*>. Valid flags are defined in <*sys/module_khelp.h*>. The HELPER_NEEDS_OSD flag should be set in the *h_flags* field if the **khelp** module requires persistent per-object data storage. There is no programmatic way (yet) to check if a **khelp** class provides the ability for **khelp** modules to associate persistent per-object data, so a manual check is required.

The **KHELP_DECLARE_MOD**() and **KHELP_DECLARE_MOD_UMA**() macros provide convenient wrappers around the DECLARE_MODULE(9) macro, and are used to register a **khelp** module with the **khelp** framework. **KHELP_DECLARE_MOD_UMA**() should only be used by modules which require the use of persistent per-object storage i.e. modules which set the HELPER_NEEDS_OSD flag in their *struct helper*'s *h_flags* field.

The first four arguments common to both macros are as follows. The *hname* argument specifies the unique ascii(7) name for the **khelp** module. It should be no longer than HELPER_NAME_MAXLEN-1 characters in length. The *hdata* argument is a pointer to the module's *struct helper*. The *hhooks* argument points to a static array of *struct hookinfo* structures. The array should contain a *struct hookinfo* for each hhook(9) point the module wishes to hook, even when using the same hook function multiple times for different hhook(9) points. The *version* argument specifies a version number for the module which will be passed to MODULE_VERSION(9). The **KHELP_DECLARE_MOD_UMA**()

macro takes the additional *ctor* and *dtor* arguments, which specify optional uma(9) constructor and destructor functions. NULL should be passed where the functionality is not required.

The **khelp_get_id**() function returns the numeric identifier for the **khelp** module with name *hname*.

The **khelp_get_osd**() function is used to obtain the per-object data pointer for a specified **khelp** module. The *hosd* argument is a pointer to the underlying subsystem object's *struct osd*. This is provided by the hhook(9) framework when calling into a **khelp** module's hook function. The *id* argument specifies the numeric identifier for the **khelp** module to extract the data pointer from *hosd* for. The *id* is obtained using the **khelp_get_id**() function.

The **khelp_add_hhook**() and **khelp_remove_hhook**() functions allow a **khelp** module to dynamically hook/unhook hhook(9) points at run time. The *hki* argument specifies a pointer to a *struct hookinfo* which encapsulates the required information about the hhook(9) point and hook function being manipulated. The HHOOK_WAITOK flag may be passed in via the *flags* argument of **khelp_add_hhook**() if malloc(9) is allowed to sleep waiting for memory to become available.

Integrating Khelp Into a Kernel Subsystem

Most of the work required to allow **khelp** modules to do useful things relates to defining and instantiating suitable hhook(9) points for **khelp** modules to hook into. The only additional decision a subsystem needs to make is whether it wants to allow **khelp** modules to associate persistent per-object data. Providing support for persistent data storage can allow **khelp** modules to perform more complex functionality which may be desirable. Subsystems which want to allow Khelp modules to associate persistent per-object data with one of the subsystem's data structures need to make the following two key changes:

- Embed a *struct osd* pointer in the structure definition for the object.

- Add calls to **khelp_init_osd**() and **khelp_destroy_osd**() to the subsystem code paths which are responsible for respectively initialising and destroying the object.

The **khelp_init_osd**() function initialises the per-object data storage for all currently loaded **khelp** modules of appropriate classes which have set the HELPER_NEEDS_OSD flag in their *h_flags* field. The *classes* argument specifies a bitmask of **khelp** classes which this subsystem associates with. If a **khelp** module matches any of the classes in the bitmask, that module will be associated with the object. The *hosd* argument specifies the pointer to the object's *struct osd* which will be used to provide the persistent storage for use by **khelp** modules.

The **khelp_destroy_osd**() function frees all memory that was associated with an object's *struct osd* by a previous call to **khelp_init_osd**(). The *hosd* argument specifies the pointer to the object's *struct osd*

which will be purged in preparation for destruction.

IMPLEMENTATION NOTES

khelp modules are protected from being prematurely unloaded by a reference count. The count is incremented each time a subsystem calls **khelp_init_osd**() causing persistent storage to be allocated for the module, and decremented for each corresponding call to **khelp_destroy_osd**(). Only when a module's reference count has dropped to zero can the module be unloaded.

RETURN VALUES

The **khelp_init_osd**() function returns zero if no errors occurred. It returns ENOMEM if a **khelp** module which requires per-object storage fails to allocate the necessary memory.

The **khelp_destroy_osd**() function only returns zero to indicate that no errors occurred.

The **khelp_get_id**() function returns the unique numeric identifier for the registered **khelp** module with name *hname*. It return -1 if no module with the specified name is currently registered.

The **khelp_get_osd**() function returns the pointer to the **khelp** module's persistent object storage memory. If the module identified by *id* does not have persistent object storage registered with the object's *hosd struct osd*, NULL is returned.

The **khelp_add_hhook**() function returns zero if no errors occurred. It returns ENOENT if it could not find the requested hhook(9) point. It returns ENOMEM if malloc(9) failed to allocate memory. It returns EEXIST if attempting to register the same hook function more than once for the same hhook(9) point.

The **khelp_remove_hhook**() function returns zero if no errors occurred. It returns ENOENT if it could not find the requested hhook(9) point.

EXAMPLES

A well commented example Khelp module can be found at: */usr/share/examples/kld/khelp/h_example.c*

The Enhanced Round Trip Time (ERTT) h_ertt(4) **khelp** module provides a more complex example of what is possible.

SEE ALSO

h_ertt(4), hhook(9), osd(9)

ACKNOWLEDGEMENTS

Development and testing of this software were made possible in part by grants from the FreeBSD

Foundation and Cisco University Research Program Fund at Community Foundation Silicon Valley.

HISTORY

The **khelp** kernel helper framework first appeared in FreeBSD 9.0.

The **khelp** framework was first released in 2010 by Lawrence Stewart whilst studying at Swinburne University of Technology's Centre for Advanced Internet Architectures, Melbourne, Australia. More details are available at:

http://caia.swin.edu.au/urp/newtcp/

AUTHORS

The **khelp** framework was written by Lawrence Stewart <*lstewart@FreeBSD.org*>.

This manual page was written by David Hayes <*david.hayes@ieee.org*> and Lawrence Stewart <*lstewart@FreeBSD.org*>.