NAME

hhook, **hhook_head_register**, **hhook_head_deregister**, **hhook_head_deregister_lookup**, **hhook_run_hooks**, **HHOOKS_RUN_IF**, **HHOOKS_RUN_LOOKUP_IF** - Helper Hook Framework

SYNOPSIS

#include <sys/hhook.h>

typedef int
(*hhook_func_t)(*int32_t hhook_type, int32_t hhook_id, void *udata, void *ctx_data, void *hdata, struct osd *hosd*);

int hhook_head_register(*int32_t hhook_type, int32_t hhook_id, struct hhook_head **hhh, uint32_t flags*);

int hhook_head_deregister(*struct hhook_head *hhh*);

int hhook_head_deregister_lookup(*int32_t hhook_type, int32_t hhook_id*);

void hhook_run_hooks(*struct hhook_head *hhh, void *ctx_data, struct osd *hosd*);

HHOOKS_RUN_IF(*hhh, ctx_data, hosd*);

HHOOKS_RUN_LOOKUP_IF(*hhook_type, hhook_id, ctx_data, hosd*);

DESCRIPTION

hhook provides a framework for managing and running arbitrary hook functions at defined hook points within the kernel. The KPI was inspired by pfil(9), and in many respects can be thought of as a more generic superset of pfil.

The khelp(9) and **hhook** frameworks are tightly integrated. Khelp is responsible for registering and deregistering Khelp module hook functions with **hhook** points. The KPI functions used by khelp(9) to do this are not documented here as they are not relevant to consumers wishing to instantiate hook points.

Information for Khelp Module Implementors

Khelp modules indirectly interact with **hhook** by defining appropriate hook functions for insertion into hook points. Hook functions must conform to the *hhook_func_t* function pointer declaration outlined in the *SYNOPSIS*.

The *hhook_type* and *hhook_id* arguments identify the hook point which has called into the hook function. These are useful when a single hook function is registered for multiple hook points and wants

to know which hook point has called into it. <*sys/hhook.h*> lists available *hhook_type* defines and subsystems which export hook points are responsible for defining the *hhook_id* value in appropriate header files.

The *udata* argument will be passed to the hook function if it was specified in the *struct hookinfo* at hook registration time.

The *ctx_data* argument contains context specific data from the hook point call site. The data type passed is subsystem dependent.

The *hdata* argument is a pointer to the persistent per-object storage allocated for use by the module if required. The pointer will only ever be NULL if the module did not request per-object storage.

The *hosd* argument can be used with the khelp(9) framework's **khelp_get_osd**() function to access data belonging to a different Khelp module.

Khelp modules instruct the Khelp framework to register their hook functions with **hhook** points by creating a *struct hookinfo* per hook point, which contains the following members:

```
struct hookinfo {
        hhook_func_t    hook_func;
        struct helper   *hook_helper;
        void            *hook_udata;
        int32_t         hook_id;
        int32_t         hook_type;
};
```

Khelp modules are responsible for setting all members of the struct except *hook_helper* which is handled by the Khelp framework.

Creating and Managing Hook Points

Kernel subsystems that wish to provide **hhook** points typically need to make four and possibly five key changes to their implementation:

- Define a list of *hhook_id* mappings in an appropriate subsystem header.
- Register each hook point with the **hhook_head_register**() function during initialisation of the subsystem.
- Select or create a standardised data type to pass to hook functions as contextual data.

- Add a call to **HHOOKS_RUN_IF**() or **HHOOKS_RUN_IF_LOOKUP**() at the point in the subsystem's code where the hook point should be executed.

- If the subsystem can be dynamically added/removed at runtime, each hook point registered with the **hhook_head_register**() function when the subsystem was initialised needs to be deregistered with the **hhook_head_deregister**() or **hhook_head_deregister_lookup**() functions when the subsystem is being deinitialised prior to removal.

The **hhook_head_register**() function registers a hook point with the **hhook** framework. The *hook_type* argument defines the high level type for the hook point. Valid types are defined in <*sys/hhook.h*> and new types should be added as required. The *hook_id* argument specifies a unique, subsystem specific identifier for the hook point. The *hhh* argument will, if not NULL, be used to store a reference to the *struct hhook_head* created as part of the registration process. Subsystems will generally want to store a local copy of the *struct hhook_head* so that they can use the **HHOOKS_RUN_IF**() macro to instantiate hook points. The HHOOK_WAITOK flag may be passed in via the *flags* argument if malloc(9) is allowed to sleep waiting for memory to become available. If the hook point is within a virtualised subsystem (e.g. the network stack), the HHOOK_HEADISINVNET flag should be passed in via the *flags* argument so that the *struct hhook_head* created during the registration process will be added to a virtualised list.

The **hhook_head_deregister**() function deregisters a previously registered hook point from the **hhook** framework. The *hhh* argument is the pointer to the *struct hhook_head* returned by **hhoook_head_register**() when the hook point was registered.

The **hhook_head_deregister_lookup**() function can be used instead of **hhook_head_deregister**() in situations where the caller does not have a cached copy of the *struct hhook_head* and wants to deregister a hook point using the appropriate *hook_type* and *hook_id* identifiers instead.

The **hhook_run_hooks**() function should normally not be called directly and should instead be called indirectly via the **HHOOKS_RUN_IF**() macro. However, there may be circumstances where it is preferable to call the function directly, and so it is documented here for completeness. The *hhh* argument references the **hhook** point to call all registered hook functions for. The *ctx_data* argument specifies a pointer to the contextual hook point data to pass into the hook functions. The *hosd* argument should be the pointer to the appropriate object's *struct osd* if the subsystem provides the ability for Khelp modules to associate per-object data. Subsystems which do not should pass NULL.

The **HHOOKS_RUN_IF**() macro is the preferred way to implement hook points. It only calls the **hhook_run_hooks**() function if at least one hook function is registered for the hook point. By checking for registered hook functions, the macro minimises the cost associated with adding hook points to frequently used code paths by reducing to a simple if test in the common case where no hook functions

are registered. The arguments are as described for the **hhook_run_hooks**() function.

The **HHOOKS_RUN_IF_LOOKUP**() macro performs the same function as the **HHOOKS_RUN_IF**() macro, but performs an additional step to look up the *struct hhook_head* for the specified *hook_type* and *hook_id* identifiers. It should not be used except in code paths which are infrequently executed because of the reference counting overhead associated with the look up.

IMPLEMENTATION NOTES

Each *struct hhook_head* protects its internal list of hook functions with a rmlock(9). Therefore, anytime **hhook_run_hooks**() is called directly or indirectly via the **HHOOKS_RUN_IF**() or **HHOOKS_RUN_IF_LOOKUP**() macros, a non-sleepable read lock will be acquired and held across the calls to all registered hook functions.

RETURN VALUES

hhook_head_register() returns 0 if no errors occurred. It returns EEXIST if a hook point with the same *hook_type* and *hook_id* is already registered. It returns EINVAL if the HHOOK_HEADISINVNET flag is not set in *flags* because the implementation does not yet support hook points in non-virtualised subsystems (see the *BUGS* section for details). It returns ENOMEM if malloc(9) failed to allocate memory for the new *struct hhook_head*.

hhook_head_deregister() and **hhook_head_deregister_lookup**() return 0 if no errors occurred. They return ENOENT if *hhh* is NULL. They return EBUSY if the reference count of *hhh* is greater than one.

EXAMPLES

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

The tcp(4) implementation provides two **hhook** points which are called for packets sent/received when a connection is in the established phase. Search for HHOOK in the following files: *sys/netinet/tcp_var.h*, *sys/netinet/tcp_input.c*, *sys/netinet/tcp_output.c* and *sys/netinet/tcp_subr.c*.

SEE ALSO

khelp(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 **hhook** framework first appeared in FreeBSD 9.0.

The **hhook** 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 **hhook** 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*>.