kREproxy

0.1

Generated by Doxygen 1.5.8

Fri Feb 20 19:46:05 2009

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 kre_config_entry Struct Reference

Configuration entry.

`#include <kre_config.h>`

Data Fields

- struct list_head list
  Pointer to next/prev entry.
- char * req
  Configuration URI.
- size_t req_size
  Size of req.
- char * host
  Configuration host.
- size_t host_size
  Size of host.
- u8 dst_host [4]
  Destination IP addres.
- int dst_port
  Destination port number.

3.1.1 Detailed Description

Configuration entry.

3.1.2 Field Documentation

3.1.2.1 u8 kre_config_entry::dst_host[4]

Destination IP addres.

3.1.2.2 int kre_config_entry::dst_port

Destination port number.

3.1.2.3 char* kre_config_entry::host

Configuration host.

3.1.2.4 size_t kre_config_entry::host_size

Size of host.

3.1.2.5 struct list_head kre_config_entry::list [read]

Pointer to next/prev entry.

3.1.2.6 char* kre_config_entry::req

Configuration URI.

3.1.2.7 size_t kre_config_entry::req_size

Size of req.

The documentation for this struct was generated from the following file:

- kre_config.h

3.2 kre_http_message_header Struct Reference

Message header.

`#include <kre_http.h>`

Data Fields

- char field_value [1000]
  Field value.
- size_t fv_size
  Field value size.

3.2.1 Detailed Description

Message header.

The same as in HTTP protocol meaning.

3.2.2 Field Documentation

3.2.2.1 char kre_http_message_header::field_value[1000]

Field value.

3.2.2.2 size_t kre_http_message_header::fv_size

Field value size.

The documentation for this struct was generated from the following file:

- kre_http.h

3.3 kre_http_request Struct Reference

kREproxy HTTP request representation.

`#include <kre_http.h>`

Data Fields

- int method
  method / HTTP/1.1 value
- struct kre_http_message_header uri
  METHOD uri HTTP/1.1 value.
- struct kre_http_message_header host
  Host: HTTP header value.
- size_t content_length
  Content-Length: HTTP header value.
- int state
  State of request for examp.

3.3.1 Detailed Description

kREproxy HTTP request representation.

3.3.2 Field Documentation

3.3.2.1 int kre_http_request::state

State of request for examp.

KRE_HTTP_BAD_REQUEST, ...NOT_IMPLEMENTED,...

The documentation for this struct was generated from the following file:

- kre_http.h

3.4 kre_thread Struct Reference

kREproxy thread representation

`#include <kre_threads.h>`

Data Fields

- struct task_struct * task
  Struct of actual kernel thread.
- int(* fun)(void *)
  Thread function with pointer to this struct.
- void * **data**
- int state
  State can be KRE_INITIALIZED, KRE_RUNNING .
- struct semaphore state_sem
  SEMAPHOR for change state.
- struct completion c
  Info about completion of this thread this use instaid of 'state' when kre_thread struct must be clean.
- struct socket * csock
  Accepted client socket.
- struct socket * dstsock
  Proxy destination server socket.

3.4.1 Detailed Description

kREproxy thread representation

3.4.2 Field Documentation

3.4.2.1 int(* kre_thread::fun)(void *)

Thread function with pointer to this struct.

3.4.2.2 int kre_thread::state

State can be KRE_INITIALIZED, KRE_RUNNING .

..

3.4.2.3 struct semaphore kre_thread::state_sem [read]

SEMAPHOR for change state.

3.4.2.4 struct task_struct* kre_thread::task [read]

Struct of actual kernel thread.

The documentation for this struct was generated from the following file:

- kre_threads.h

Chapter 4

File Documentation

4.1 kre_config.c File Reference

kREproxy configuration interface implementation.

```
#include "kre_config.h"
```

Functions

- void kre_config_register (kre_config *c)

  kREproxy configuration registration

- int kre_config_unregister (kre_config *c)

  kREproxy configuration unregistration

- struct kre_config_entry * kre_config_new_entry (const char *req, size_t req_size, const char *host, size_t host_size, u8 dst_host[ ], int dst_port)

  Creating new config entry.

- int kre_config_add_entry_tail (struct kre_config_entry *new, kre_config *conf)

  Adding new configuration entry in tail of configuration list.

- int kre_config_add_entry (kre_config *c, const char *uri, size_t us, const char *host, size_t hs, u8 dst_host[ ], int dst_port)

  Creating and adding configuration entry.

- int kre_config_add_default_entry (kre_config *c, u8 dst_host[ ], int dst_port)

  Adding default entry.

- int kre_config_is_uri_equal (const char *u1, size_t s1, const char *u2, size_t s2)

  Equal test of two uri from HTTP request and from configuration.

- int kre_config_is_host_equal (const char *h1, size_t s1, const char *h2, size_t s2)

  Equal test of two hosts from HTTP request and configuation.

- struct kre_config_entry * kre_config_find (kre_config *c, const char *uri, size_t us, const char *host, size_t hs)

  Find configuration structure witch tell kreproxy where pass the request from client.

4.1.1 Detailed Description

kREproxy configuration interface implementation.

4.1.2 Function Documentation

4.1.2.1 int kre_config_add_default_entry (kre_config * *c*, u8 *dst_host*[ ], int *dst_port*)

Adding default entry.

This is default destination host to service requst which not match to any other configuration entry. YOU HAVE TO REMEMBER THAT THIS ENTRY MUST BE SET AS LAST CONFIGURATION ENTRY.

Parameters:

dst_host Destination host.

dst_port Destination port.

Returns:

error code or 0

4.1.2.2 int kre_config_add_entry (kre_config * *c*, const char * *uri*, size_t *us*, const char * *host*, size_t *hs*, u8 *dst_host*[ ], int *dst_port*)

Creating and adding configuration entry.

This is main configuration adding function which end user should use.

Parameters:

c Pointer to main kREproxy configuration.

uri HTTP URI.

us Size of URI.

host HTTP host.

hs Size of host.

dst_host Destination host.

dst_port Destination port.

Returns:

Error code or 0.

4.1.2.3 int kre_config_add_entry_tail (struct kre_config_entry * *new*, kre_config * *conf*)

Adding new configuration entry in tail of configuration list.

Parameters:

new New added entry.

conf Actual configuration - place to add new entry.

Returns:

Error code or 0.

4.1.2.4 struct kre_config_entry* kre_config_find (kre_config * *c*, const char * *uri*, size_t *us*, const char * *host*, size_t *hs*) [read]

Find configuration structure witch tell kreproxy where pass the request from client.

Parameters:

c Pointer to main kREproxy configuration.

uri URI from client HTTP request.

us Size of URI.

host Host from client HTTP request.

hs Size of host.

Returns:

Pointer to right configuration entry.

4.1.2.5 int kre_config_is_host_equal (const char * *h1*, size_t *s1*, const char * *h2*, size_t *s2*)

Equal test of two hosts from HTTP request and configuation.

This is simple equal test. If all of chars are equal this function return 1 in other hand 0.

Parameters:

h1 Host 1 to equal test, which have to come from kre configuration.

s1 Size of host 1.

h2 Host 2 to equal test, which have to come from HTTP request.

s2 Size of host 2.

Returns:

0 if not equal or 1 if equal.

4.1.2.6 int kre_config_is_uri_equal (const char * *u1*, size_t *s1*, const char * *u2*, size_t *s2*)

Equal test of two uri from HTTP request and from configuration.

This is specific kreproxy comparison this can interpret asterisk ('*') symbol on The end of uri as string of any char or zero chars.

Parameters:

u1 URI 1 to equal test, which have to come from kre configuration.

s1 Size of uri 1.

u2 URI 2 to equal test, which have to come from HTTP request.

s2 Size of uri 2.

Returns:

0 if not equal, if equal 1.

4.1.2.7 struct kre_config_entry* kre_config_new_entry (const char * *req*, size_t *req_size*, const char * *host*, size_t *host_size*, u8 *dst_host*[ ], int *dst_port*) [read]

Creating new config entry.

This function create new configuration entry allocate memory for new entry copy data and return address of this entry.

Parameters:

req Http GET/POST request.

host Http Host: entry.

dst_host Destination address IP.

dst_port Destination port.

Returns:

Pointer to new allocated entry.

4.1.2.8 void kre_config_register (kre_config *)

kREproxy configuration registration

Parameters:

Pointer to main configuration structure.

4.1.2.9 int kre_config_unregister (kre_config *)

kREproxy configuration unregistration

Parameters:

Pointer to main configuration structure.

Returns:

error output or 0 if everythig ok

4.2 kre_config.h File Reference

kREproxy configuration interface

```
#include <linux/types.h>
#include <linux/slab.h>
#include "kre_list.h"
#include "kre_debug.h"
#include "kre_options.h"
```

Data Structures

- struct kre_config_entry
  Configuration entry.

Typedefs

- typedef struct list_head kre_config
  Main kREproxy configuration structure.

Functions

- void kre_config_register (kre_config *)
  kREproxy configuration registration
- int kre_config_unregister (kre_config *)
  kREproxy configuration unregistration
- struct kre_config_entry * kre_config_new_entry (const char *req, size_t req_size, const char *host, size_t host_size, u8 dst_host[ ], int dst_port)
  Creating new config entry.
- int kre_config_add_entry_tail (struct kre_config_entry *new, kre_config *conf)
  Adding new configuration entry in tail of configuration list.
- int kre_config_add_entry (kre_config *c, const char *uri, size_t us, const char *host, size_t hs, u8 dst_host[ ], int dst_port)
  Creating and adding configuration entry.
- int kre_config_add_default_entry (kre_config *c, u8 dst_host[ ], int dst_port)
  Adding default entry.
- int kre_config_is_uri_equal (const char *u1, size_t s1, const char *u2, size_t s2)
  Equal test of two uri from HTTP request and from configuration.
- int kre_config_is_host_equal (const char *h1, size_t s1, const char *h2, size_t s2)

Equal test of two hosts from HTTP request and configuation.

- struct kre_config_entry * kre_config_find (kre_config *c, const char *uri, size_t us, const char *host, size_t hs)

  Find configuration structure witch tell kreproxy where pass the request from client.

4.2.1 Detailed Description

kREproxy configuration interface

4.2.2 Typedef Documentation

4.2.2.1 typedef struct list_head kre_config

Main kREproxy configuration structure.

This should be declared once in global scope.

4.2.3 Function Documentation

4.2.3.1 int kre_config_add_default_entry (kre_config * *c*, u8 *dst_host*[ ], int *dst_port*)

Adding default entry.

This is default destination host to service requst which not match to any other configuration entry. YOU HAVE TO REMEMBER THAT THIS ENTRY MUST BE SET AS LAST CONFIGURATION ENTRY.

Parameters:

dst_host Destination host.

dst_port Destination port.

Returns:

error code or 0

4.2.3.2 int kre_config_add_entry (kre_config * *c*, const char * *uri*, size_t *us*, const char * *host*, size_t *hs*, u8 *dst_host*[ ], int *dst_port*)

Creating and adding configuration entry.

This is main configuration adding function which end user should use.

Parameters:

c Pointer to main kREproxy configuration.

uri HTTP URI.

us Size of URI.

host HTTP host.

hs Size of host.

dst_host Destination host.
dst_port Destination port.

Returns:

Error code or 0.

4.2.3.3 int kre_config_add_entry_tail (struct kre_config_entry * *new*, kre_config * *conf*)

Adding new configuration entry in tail of configuration list.

Parameters:

new New added entry.
conf Actual configuration - place to add new entry.

Returns:

Error code or 0.

4.2.3.4 struct kre_config_entry* kre_config_find (kre_config * *c*, const char * *uri*, size_t *us*, const char * *host*, size_t *hs*) [read]

Find configuration structure witch tell kreproxy where pass the request from client.

Parameters:

c Pointer to main kREproxy configuration.
uri URI from client HTTP request.
us Size of URI.
host Host from client HTTP request.
hs Size of host.

Returns:

Pointer to right configuration entry.

4.2.3.5 int kre_config_is_host_equal (const char * *h1*, size_t *s1*, const char * *h2*, size_t *s2*)

Equal test of two hosts from HTTP request and configuation.

This is simple equal test. If all of chars are equal this function return 1 in other hand 0.

Parameters:

h1 Host 1 to equal test, which have to come from kre configuration.
s1 Size of host 1.
h2 Host 2 to equal test, which have to come from HTTP request.
s2 Size of host 2.

Returns:

0 if not equal or 1 if equal.

4.2.3.6 int kre_config_is_uri_equal (const char * *u1*, size_t *s1*, const char * *u2*, size_t *s2*)

Equal test of two uri from HTTP request and from configuration.

This is specific kreproxy comparison this can interpret asterisk ('*') symbol on The end of uri as string of any char or zero chars.

Parameters:

u1 URI 1 to equal test, which have to come from kre configuration.

s1 Size of uri 1.

u2 URI 2 to equal test, which have to come from HTTP request.

s2 Size of uri 2.

Returns:

0 if not equal, if equal 1.

4.2.3.7 struct kre_config_entry* kre_config_new_entry (const char * *req*, size_t *req_size*, const char * *host*, size_t *host_size*, u8 *dst_host*[ ], int *dst_port*) [read]

Creating new config entry.

This function create new configuration entry allocate memory for new entry copy data and return address of this entry.

Parameters:

req Http GET/POST request.

host Http Host: entry.

dst_host Destination address IP.

dst_port Destination port.

Returns:

Pointer to new allocated entry.

4.2.3.8 void kre_config_register (kre_config *)

kREproxy configuration registration

Parameters:

Pointer to main configuration structure.

4.2.3.9 int kre_config_unregister (kre_config *)

kREproxy configuration unregistration

Parameters:

Pointer to main configuration structure.

Returns:

error output or 0 if everythig ok

4.3 kre_debug.h File Reference

kREproxy debuging management.

Defines

- #define **KRE_DBG**(format, args...) printk(KERN_DEBUG "kre-error: " format, ## args)
- #define **KRE_DBG_INF**(format, args...)

4.3.1 Detailed Description

kREproxy debuging management.

4.4 kre_http.c File Reference

kREproxy HTTP management interface implementation

`#include "kre_http.h"`

Functions

- int kre_http_get_request (char *bu, size_t s, struct kre_http_request *req, struct kre_thread *thr)
  Get HTTP request from client to buffer bu and return count of received bytes or error or should stop flag.
- int kre_http_transfer_response (char *bu, size_t s, struct kre_thread *thr)
  Transfer response from dst host to client host using buffer bu.
- void kre_http_set_response (char *bu, size_t size, int code)
  Build response in bu based on http_error number.
- void kre_http_get_meth_from_buf (struct kre_http_request *req, char *b, size_t size)
  Get HTTP method from b parameter.
- void kre_http_get_content_length (struct kre_http_request *req, char *b, size_t size)
  Get HTTP Content-Length if exsists value.
- void kre_http_get_uri (struct kre_http_request *req, char *b, size_t size)
  Get URI from HTTP request (from buffer b).
- void kre_http_get_host (struct kre_http_request *req, char *b, size_t size)
  Get Host from HTTP request b.
- int kre_http_get_req_from_buf (struct kre_http_request *req, char *bu, size_t size)
  Get all needed field from request buf and save to req structure.

4.4.1 Detailed Description

kREproxy HTTP management interface implementation

4.4.2 Function Documentation

4.4.2.1 void kre_http_get_content_length (struct kre_http_request * *req*, char * *b*, size_t *size*)

Get HTTP Content-Length if exsists value.

Parameters:

req Place where contetn length value is saved if not exsixts 0 value is saved.

b Buffor with http request.

size Size of buffer.

4.4.2.2 void kre_http_get_host (struct kre_http_request * *req*, char * *b*, size_t *size*)

Get Host from HTTP request b.

Parameters:

req Place to save host.

b Buffer with request.

size Size of buffer b.

4.4.2.3 void kre_http_get_meth_from_buf (struct kre_http_request * *req*, char * *b*, size_t *size*)

Get HTTP method from b parameter.

Parameters:

req Place to save http method.

b Buffor with http request.

size Size of buffer.

4.4.2.4 int kre_http_get_req_from_buf (struct kre_http_request * *req*, char * *bu*, size_t *size*)

Get all needed field from request buf and save to req structure.

Parameters:

req Place to save request field.

bu Source request buffor.

size Size of buffer.

Returns:

0 if everything is ok or -EINVAL, -1, or req->state (KRE_HTTP_BAD_REQUEST, ...).

4.4.2.5 int kre_http_get_request (char * *bu*, size_t *s*, struct kre_http_request * *req*, struct kre_thread * *cur*)

Get HTTP request from client to buffer bu and return count of received bytes or error or should stop flag.

Parameters:

bu Buffer where http request will go.

s Size of buffer.

req Place to save HTTP request in specific kre format.

cur Curent kre thread with csock.

Returns:

-KRE_HTTP_ERROR | -KRE_HTTP_SHOULD_STOP | count of received bytes.

4.4.2.6 void kre_http_get_uri (struct kre_http_request * *req*, char * *b*, size_t *size*)

Get URI from HTTP request (from buffer b).

Parameters:

req Place to save uri.

b Buffer with request.

size Size of buffer.

4.4.2.7 void kre_http_set_response (char * *bu*, size_t *size*, int *code*)

Build response in bu based on http_error number.

Parameters:

bu Buffer where new response is build.

size Max size of buffer.

code KRE_HTTP_(response code) for example KRE_HTTP_BAD_REQUEST, KRE_HTTP_-INTERNAL_SERVER_ERROR, ... , KRE_HTTP_SERVER_IS_BUSY,..

4.4.2.8 int kre_http_transfer_response (char * *bu*, size_t *s*, struct kre_thread * *thr*)

Transfer response from dst host to client host using buffer bu.

Parameters:

bu Buffer using during transfering.

s Size of buffer bu.

thr Thread structure with alocated and connecting dstsock and csock socket.

Returns:

Error code or 0.

4.5 kre_http.h File Reference

kREproxy HTTP protocol service

```
#include "kre_threads.h"
#include "kre_network.h"
#include "kre_utils.h"
```

Data Structures

- struct kre_http_message_header

  Message header.

- struct kre_http_request

  kREproxy HTTP request representation.

Defines

- #define **KRE_SERVER_NAME** "kREproxy (author: Marcin Tomasik)"
- #define **KRE_SERVER_NAME_SIZE** 33
- #define **KRE_HTTP_REQ_BUF_SIZE** 2048
- #define **KRE_HTTP_MAX_FIELD_SIZE** 100
- #define **KRE_HTTP_MAX_FIELD_VALUE_SIZE** 1000
- #define **KRE_HTTP_MAX_REQUEST_LINE_SIZE** 1000
- #define **KRE_HTTP_ERROR** 1
- #define **KRE_HTTP_SHOULD_STOP** 2
- #define **KRE_HTTP_OK** 200
- #define **KRE_HTTP_NO_CONTENT** 204
- #define **KRE_HTTP_BAD_REQUEST** 400
- #define **KRE_HTTP_NOT_FOUND** 404
- #define **KRE_HTTP_METHOD_NOT_ALLOWD** 405
- #define **KRE_HTTP_REQ_ENTITY_TOO_LARGE** 413
- #define **KRE_HTTP_REQ_URI_TOO_LARGE** 414
- #define **KRE_HTTP_DST_UNRECHABLE** 900
- #define **KRE_HTTP_SERVER_IS_BUSY** 901
- #define **KRE_HTTP_INTERNAL_SERVER_ERROR** 500
- #define **KRE_HTTP_NOT_IMPLEMENTED** 501
- #define **KRE_HTTP_SERVICE_UNAVALIBLE** 503
- #define **KRE_HTTP_METH_GET** 0
- #define **KRE_HTTP_METH_POST** 1
- #define **KRE_HTTP_METH_HEAD** 2
- #define **KRE_HTTP_METH_OPTIONS** 3
- #define **KRE_HTTP_METH_PUT** 4
- #define **KRE_HTTP_METH_DELETE** 5
- #define **KRE_HTTP_METH_TRACE** 6
- #define **KRE_HTTP_METH_CONNECT** 7
- #define **KRE_HTTP_METH_NOT_IMPLEMENTED** 8

Functions

- void kre_http_get_meth_from_buf (struct kre_http_request *req, char *b, size_t size)
  Get HTTP method from b parameter.
- void kre_http_get_content_length (struct kre_http_request *req, char *b, size_t size)
  Get HTTP Content-Length if exsists value.
- void kre_http_get_uri (struct kre_http_request *req, char *b, size_t size)
  Get URI from HTTP request (from buffer b).
- void kre_http_get_host (struct kre_http_request *req, char *b, size_t size)
  Get Host from HTTP request b.
- int kre_http_get_req_from_buf (struct kre_http_request *req, char *bu, size_t size)
  Get all needed field from request buf and save to req structure.
- int kre_http_get_request (char *bu, size_t s, struct kre_http_request *req, struct kre_thread *cur)
  Get HTTP request from client to buffer bu and return count of received bytes or error or should stop flag.
- int kre_http_transfer_response (char *bu, size_t s, struct kre_thread *thr)
  Transfer response from dst host to client host using buffer bu.
- void kre_http_set_response (char *bu, size_t size, int code)
  Build response in bu based on http_error number.

4.5.1 Detailed Description

kREproxy HTTP protocol service

4.5.2 Function Documentation

4.5.2.1 void kre_http_get_content_length (struct kre_http_request * *req*, char * *b*, size_t *size*)

Get HTTP Content-Length if exsists value.

Parameters:

req Place where contetn length value is saved if not exsixts 0 value is saved.

b Buffor with http request.

size Size of buffer.

4.5.2.2 void kre_http_get_host (struct kre_http_request * *req*, char * *b*, size_t *size*)

Get Host from HTTP request b.

Parameters:

req Place to save host.

b Buffer with request.
size Size of buffer b.

4.5.2.3 void kre_http_get_meth_from_buf (struct kre_http_request * *req*, char * *b*, size_t *size*)

Get HTTP method from b parameter.

Parameters:

req Place to save http method.
b Buffor with http request.
size Size of buffer.

4.5.2.4 int kre_http_get_req_from_buf (struct kre_http_request * *req*, char * *bu*, size_t *size*)

Get all needed field from request buf and save to req structure.

Parameters:

req Place to save request field.
bu Source request buffor.
size Size of buffer.

Returns:

0 if everything is ok or -EINVAL, -1, or req->state (KRE_HTTP_BAD_REQUEST, ...).

4.5.2.5 int kre_http_get_request (char * *bu*, size_t *s*, struct kre_http_request * *req*, struct kre_thread * *cur*)

Get HTTP request from client to buffer bu and return count of received bytes or error or should stop flag.

Parameters:

bu Buffer where http request will go.
s Size of buffer.
req Place to save HTTP request in specific kre format.
cur Curent kre thread with csock.

Returns:

-KRE_HTTP_ERROR | -KRE_HTTP_SHOULD_STOP | count of received bytes.

4.5.2.6 void kre_http_get_uri (struct kre_http_request * *req*, char * *b*, size_t *size*)

Get URI from HTTP request (from buffer b).

Parameters:

req Place to save uri.
b Buffer with request.
size Size of buffer.

4.5.2.7 void kre_http_set_response (char * *bu*, size_t *size*, int *code*)

Build response in bu based on http_error number.

Parameters:

bu Buffer where new response is build.

size Max size of buffer.

code KRE_HTTP_(response code) for example KRE_HTTP_BAD_REQUEST, KRE_HTTP_-INTERNAL_SERVER_ERROR, ... , KRE_HTTP_SERVER_IS_BUSY,..

4.5.2.8 int kre_http_transfer_response (char * *bu*, size_t *s*, struct kre_thread * *thr*)

Transfer response from dst host to client host using buffer bu.

Parameters:

bu Buffer using during transfering.

s Size of buffer bu.

thr Thread structure with alocated and connecting dstsock and csock socket.

Returns:

Error code or 0.

4.6 kre_list.c File Reference

kREproxy List interface implementation

```
#include "kre_list.h"
```

Functions

- void kre_list_init (struct list_head *node)
  kREproxy list initialization.
- void kre_list_add_entry_tail (struct list_head *new, struct list_head *head)
  Add new list entry tail.

4.6.1 Detailed Description

kREproxy List interface implementation

4.6.2 Function Documentation

4.6.2.1 void kre_list_add_entry_tail (struct list_head * *new*, struct list_head * *list*)

Add new list entry tail.

Parameters:

new New list head entry to add.

list Destination list.

4.7 kre_list.h File Reference

kREproxy list interface

```
#include <linux/list.h>
#include "kre_debug.h"
```

Defines

- #define kre_list_head struct list_head

  kREproxy list head.

- #define kre_list_del(list_head) list_del(list_head)

  Delete entry list_head from list.

- #define kre_list_for_each(pos, head) list_for_each(pos, head)

  Macro for iteration of kre list entry.

- #define kre_list_entry(ptr, type, member) list_entry(ptr, type, member)

  Macro for geting entry structure from list Look for <linux/list.h> for more detail.

Functions

- void kre_list_init (struct list_head *)

  kREproxy list initialization.

- void kre_list_add_entry_tail (struct list_head *new, struct list_head *list)

  Add new list entry tail.

4.7.1 Detailed Description

kREproxy list interface

4.7.2 Define Documentation

4.7.2.1 #define kre_list_del(list_head) list_del(list_head)

Delete entry list_head from list.

Parameters:

list_head The entry to delete.

4.7.2.2 #define kre_list_entry(ptr, type, member) list_entry(ptr, type, member)

Macro for geting entry structure from list Look for <linux/list.h> for more detail.

Parameters:

ptr Pointer to list head.

type Type of entry.

member list head member name.

Returns:

Pointer to entry.

4.7.2.3 #define kre_list_for_each(pos, head) list_for_each(pos, head)

Macro for iteration of kre list entry.

Parameters:

pos The &struct list_head to use as a loop cursor.

head The head for your list.

4.7.3 Function Documentation

4.7.3.1 void kre_list_add_entry_tail (struct list_head * *new*, struct list_head * *list*)

Add new list entry tail.

Parameters:

new New list head entry to add.

list Destination list.

4.8 kre_main.c File Reference

Main kREproxy Linux module implementation.

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/net.h>
#include <linux/netdevice.h>
#include <linux/delay.h>
#include "kre_list.h"
#include "kre_config.h"
#include "kre_threads.h"
#include "kre_network.h"
#include "kre_http.h"
#include "kre_options.h"
```

Functions

- **MODULE_LICENSE** ("GPL")
- **MODULE_AUTHOR** ("Marcin Tomasik <marcin.tomasik.priv[onserver]gmail.com>")
- **MODULE_DESCRIPTION** ("kREproxy - simple reverse proxy [kreproxy.sourceforge.net]")
- **MODULE_VERSION** ("0.1")
- **module_init** (kre_init)
- **module_exit** (kre_exit)

4.8.1 Detailed Description

Main kREproxy Linux module implementation.

4.9 kre_network.c File Reference

kREproxy network interface implementation

```
#include "kre_network.h"
```

Functions

- void kre_net_set_local (struct sockaddr_in *a, int port)

  Set addres to 0.0.0.0 and port to port.

- void kre_net_set_addr (struct sockaddr_in *a, u8 ip[ ], int port)

  Set addres to ip and port.

- void kre_net_sock_release (struct socket **s)

  Check if not null and release socket.

- int kre_net_sock_create (struct socket **s)

  Creating (alocating) new socket.

- int kre_net_sock_setopt_reuse (struct socket *s)

  Set one of the socket options.

- int kre_net_sock_bind (struct socket *s, struct sockaddr_in *a)

  Binding addres structure with socket.

- int kre_net_sock_listen (struct socket *s)

  Turn on listening on socket.

- int kre_net_sock_accept (struct socket *srv, struct socket **cli)

  Accepting client TCP request.

- int kre_net_connect (struct socket *s, struct sockaddr_in *a)

  Connect to server with addres a using socket s.

- int kre_net_sock_receive (struct socket *s, struct sockaddr_in *a, unsigned char *buf, int len)

  Receive len bytes from client socket s.

- int kre_net_sock_send (struct socket *s, struct sockaddr_in *a, unsigned char *buf, int len)

  Sending data to client socket.

4.9.1 Detailed Description

kREproxy network interface implementation

4.9.2 Function Documentation

4.9.2.1 int kre_net_connect (struct socket * *s*, struct sockaddr_in * *a*)

Connect to server with addres a using socket s.

Parameters:

s Socket

a Addres

Returns:

error code or 0

4.9.2.2 void kre_net_set_addr (struct sockaddr_in * *a*, u8 *ip*[ ], int *port*)

Set addres to ip and port.

Parameters:

a Pointer to addres which we set.

ip IP addres to set.

port Port number to set.

4.9.2.3 void kre_net_set_local (struct sockaddr_in * *a*, int *port*)

Set addres to 0.0.0.0 and port to port.

Parameters:

a Addres to set.

port Port number.

4.9.2.4 int kre_net_sock_accept (struct socket * *srv*, struct socket ** *cli*)

Accepting client TCP request.

Parameters:

srv Pointer to server socket.

cli Pointer to pointer client socket.

4.9.2.5 int kre_net_sock_bind (struct socket * *s*, struct sockaddr_in * *a*)

Binding addres structure with socket.

Parameters:

s Socket to bind.

a Addres structure to bind.

4.9.2.6 int kre_net_sock_listen (struct socket * *s*)

Turn on listening on socket.

Parameters:

s Socket which should listen.

4.9.2.7 int kre_net_sock_receive (struct socket * *s*, struct sockaddr_in * *a*, unsigned char * *buf*, int *len*)

Receive len bytes from client socket s.

Parameters:

s Client socket.

a Client addres.

buf Buffer to handle data from client.

len Max bytes to get from client.

4.9.2.8 int kre_net_sock_send (struct socket * *s*, struct sockaddr_in * *a*, unsigned char * *buf*, int *len*)

Sending data to client socket.

Parameters:

s Client socket.

a Cut addres to send to the client.

buf Data buffer which will be send to the client.

len Count of byte from data buffer which will be send to the client.

4.9.2.9 int kre_net_sock_setopt_reuse (struct socket * *s*)

Set one of the socket options.

REUSEADDR = 1 this is not work and need recognition.

4.10 kre_network.h File Reference

kREproxy network interface.

```
#include <linux/module.h>
#include <linux/net.h>
#include <asm/uaccess.h>
#include <linux/in.h>
#include "kre_debug.h"
```

Functions

- void kre_net_set_local (struct sockaddr_in *a, int port)
  Set addres to 0.0.0.0 and port to port.
- void kre_net_set_addr (struct sockaddr_in *a, u8 ip[ ], int port)
  Set addres to ip and port.
- void kre_net_sock_release (struct socket **s)
  Check if not null and release socket.
- int kre_net_sock_create (struct socket **s)
  Creating (alocating) new socket.
- int kre_net_sock_setopt_reuse (struct socket *s)
  Set one of the socket options.
- int kre_net_sock_bind (struct socket *s, struct sockaddr_in *a)
  Binding addres structure with socket.
- int kre_net_sock_listen (struct socket *s)
  Turn on listening on socket.
- int kre_net_sock_accept (struct socket *srv, struct socket **cli)
  Accepting client TCP request.
- int kre_net_connect (struct socket *s, struct sockaddr_in *a)
  Connect to server with addres a using socket s.
- int kre_net_sock_receive (struct socket *s, struct sockaddr_in *a, unsigned char *buf, int len)
  Receive len bytes from client socket s.
- int kre_net_sock_send (struct socket *s, struct sockaddr_in *a, unsigned char *buf, int len)
  Sending data to client socket.

4.10.1 Detailed Description

kREproxy network interface.

4.10.2 Function Documentation

4.10.2.1 int kre_net_connect (struct socket * *s*, struct sockaddr_in * *a*)

Connect to server with addres a using socket s.

Parameters:

s Socket

a Addres

Returns:

error code or 0

4.10.2.2 void kre_net_set_addr (struct sockaddr_in * *a*, u8 *ip*[ ], int *port*)

Set addres to ip and port.

Parameters:

a Pointer to addres which we set.

ip IP addres to set.

port Port number to set.

4.10.2.3 void kre_net_set_local (struct sockaddr_in * *a*, int *port*)

Set addres to 0.0.0.0 and port to port.

Parameters:

a Addres to set.

port Port number.

4.10.2.4 int kre_net_sock_accept (struct socket * *srv*, struct socket ** *cli*)

Accepting client TCP request.

Parameters:

srv Pointer to server socket.

cli Pointer to pointer client socket.

4.10.2.5 int kre_net_sock_bind (struct socket * *s*, struct sockaddr_in * *a*)

Binding addres structure with socket.

Parameters:

s Socket to bind.

a Addres structure to bind.

4.10.2.6 int kre_net_sock_listen (struct socket * *s*)

Turn on listening on socket.

Parameters:

s Socket which should listen.

4.10.2.7 int kre_net_sock_receive (struct socket * *s*, struct sockaddr_in * *a*, unsigned char * *buf*, int *len*)

Receive len bytes from client socket s.

Parameters:

s Client socket.

a Client addres.

buf Buffer to handle data from client.

len Max bytes to get from client.

4.10.2.8 int kre_net_sock_send (struct socket * *s*, struct sockaddr_in * *a*, unsigned char * *buf*, int *len*)

Sending data to client socket.

Parameters:

s Client socket.

a Cut addres to send to the client.

buf Data buffer which will be send to the client.

len Count of byte from data buffer which will be send to the client.

4.10.2.9 int kre_net_sock_setopt_reuse (struct socket * *s*)

Set one of the socket options.

REUSEADDR = 1 this is not work and need recognition.

4.11 kre_options.c File Reference

kREproxy options implementations

```
#include "kre_options.h"
```

4.11.1 Detailed Description

kREproxy options implementations

4.12 kre_options.h File Reference

kREproxy options

Defines

- #define **MOD_NAME** "kREproxy"
- #define **MOD_CLIENT** "kREclient"
- #define **MOD_VER** "0.1"
- #define **KRE_LISTEN_PORT** 80

4.12.1 Detailed Description

kREproxy options

4.13 kre_threads.c File Reference

kREproxy threads management interface implementation

```
#include "kre_threads.h"
```

Functions

- void kre_threads_register_one_thread (struct kre_thread *t)

  Registration one thread.

- void kre_threads_register (kre_threads *t)

  Registration group of thread.

- int kre_threads_unregister (kre_threads *t)

  Unregistration group of thread.

- struct kre_thread * kre_threads_fff (kre_threads t)

  Find Firs Free kre_thread structure.

- int kre_threads_find_by_pid (pid_t p, kre_threads t)

  Find localization of kre_thread in kre_threads set by pid.

- struct kre_thread * kre_threads_get_by_id (int id, kre_threads t)

  Return kre thread by id.

- int kre_threads_should_work (struct kre_thread *t)

  It can tell whether kthread should work or finish all jobs.

- void kre_threads_complete (struct kre_thread *t)

  Set things, and inform other task that actual task is completed.

- int kre_threads_init (struct kre_thread *t, int(*f)(void *), void *data)

  Initialization of thread.

- int kre_threads_start (struct kre_thread *t)

  Starting the new thread.

- int kre_threads_stop (struct kre_thread *t)

  The main cause of exist this function is to kill main kreproxy server thread in clear way.

4.13.1 Detailed Description

kREproxy threads management interface implementation

4.13.2 Function Documentation

4.13.2.1 void kre_threads_complete (struct kre_thread * *t*)

Set things, and inform other task that actual task is completed.

Parameters:

t Current kre thread structure.

4.13.2.2 struct kre_thread* kre_threads_fff (kre_threads *t*) [read]

Find Firs Free kre_thread structure.

Parameters:

t Set of threads to search.

Returns:

Thread which was found or NULL.

4.13.2.3 int kre_threads_find_by_pid (pid_t *p*, kre_threads *t*)

Find localization of kre_thread in kre_threads set by pid.

Parameters:

p Function search t set looking for pid p.

t Set of kre threads.

Returns:

id of kre_thread struct in set of kre threads or -1 if not found.

4.13.2.4 struct kre_thread* kre_threads_get_by_id (int *id*, kre_threads *t*) [read]

Return kre thread by id.

Parameters:

id Identyficator returned for example by kre_threads_find_by_pid().

t Set of kre threads.

4.13.2.5 int kre_threads_init (struct kre_thread * *t*, int(*)(void *) *f*, void * *data*)

Initialization of thread.

Parameters:

t Thread to initlize.

f Thread function.

data Pointer to external thread data. NOT WORK AT NOW.

Returns:

Error report or 0 if ok.

4.13.2.6 void kre_threads_register_one_thread (struct kre_thread * *t*)

Registration one thread.

This can be used to register main kREproxy server thread.

4.13.2.7 int kre_threads_should_work (struct kre_thread * *t*)

It can tell whether kthread should work or finish all jobs.

Parameters:

t Current thread structure.

Returns:

0 - if should stop, 1 - if should work.

4.13.2.8 int kre_threads_start (struct kre_thread * *t*)

Starting the new thread.

Parameters:

t Kreporxy thread to start.

Returns:

Error report.

4.13.2.9 int kre_threads_stop (struct kre_thread * *t*)

The main cause of exist this function is to kill main kreproxy server thread in clear way.

Parameters:

t kre_thread structure with running task.

Returns:

Error output.

4.14 kre_threads.h File Reference

kREproxy threads management interface

```
#include <linux/kthread.h>
#include <linux/netdevice.h>
#include <linux/completion.h>
#include <linux/delay.h>
#include <asm/semaphore.h>
#include "kre_debug.h"
#include "kre_network.h"
```

Data Structures

- struct kre_thread

  kREproxy thread representation

Defines

- #define **KRE_NAME** "kreproxythread"
- #define **KRE_NOT_INITIALIZED** 0
- #define **KRE_INITIALIZED** 1
- #define **KRE_RUNNING** 2
- #define **KRE_SHOULD_STOP** 3
- #define **KRE_COMPLETED** 4
- #define **KRE_MAX_THREADS** 10

Typedefs

- typedef struct kre_thread kre_threads [10]

  kREproxy thread group representation This should be declared once in global scope.

Functions

- void kre_threads_register_one_thread (struct kre_thread *t)

  Registration one thread.

- void kre_threads_register (kre_threads *)

  Registration group of thread.

- int kre_threads_unregister (kre_threads *)

  Unregistration group of thread.

- int kre_threads_init (struct kre_thread *t, int(*f)(void *), void *data)

Initialization of thread.

- int kre_threads_start (struct kre_thread *t)

  Starting the new thread.

- struct kre_thread * kre_threads_fff (kre_threads t)

  Find Firs Free kre_thread structure.

- int kre_threads_find_by_pid (pid_t p, kre_threads t)

  Find localization of kre_thread in kre_threads set by pid.

- struct kre_thread * kre_threads_get_by_id (int id, kre_threads t)

  Return kre thread by id.

- void kre_threads_complete (struct kre_thread *t)

  Set things, and inform other task that actual task is completed.

- int kre_threads_should_work (struct kre_thread *t)

  It can tell whether kthread should work or finish all jobs.

- int kre_threads_stop (struct kre_thread *t)

  The main cause of exist this function is to kill main kreproxy server thread in clear way.

4.14.1 Detailed Description

kREproxy threads management interface

4.14.2 Function Documentation

4.14.2.1 void kre_threads_complete (struct kre_thread * *t*)

Set things, and inform other task that actual task is completed.

Parameters:

t Current kre thread structure.

4.14.2.2 struct kre_thread* kre_threads_fff (kre_threads *t*) [read]

Find Firs Free kre_thread structure.

Parameters:

t Set of threads to search.

Returns:

Thread which was found or NULL.

4.14.2.3 int kre_threads_find_by_pid (pid_t *p*, kre_threads *t*)

Find localization of kre_thread in kre_threads set by pid.

Parameters:

p Function search t set looking for pid p.

t Set of kre threads.

Returns:

id of kre_thread struct in set of kre threads or -1 if not found.

4.14.2.4 struct kre_thread* kre_threads_get_by_id (int *id*, kre_threads *t*) [read]

Return kre thread by id.

Parameters:

id Identyficator returned for example by kre_threads_find_by_pid().

t Set of kre threads.

4.14.2.5 int kre_threads_init (struct kre_thread * *t*, int(*)(void *) *f*, void * *data*)

Initialization of thread.

Parameters:

t Thread to initlize.

f Thread function.

data Pointer to external thread data. NOT WORK AT NOW.

Returns:

Error report or 0 if ok.

4.14.2.6 void kre_threads_register_one_thread (struct kre_thread * *t*)

Registration one thread.

This can be used to register main kREproxy server thread.

4.14.2.7 int kre_threads_should_work (struct kre_thread * *t*)

It can tell whether kthread should work or finish all jobs.

Parameters:

t Current thread structure.

Returns:

0 - if should stop, 1 - if should work.

4.14.2.8 int kre_threads_start (struct kre_thread * t)

Starting the new thread.

Parameters:

t Kreporxy thread to start.

Returns:

Error report.

4.14.2.9 int kre_threads_stop (struct kre_thread * t)

The main cause of exist this function is to kill main kreproxy server thread in clear way.

Parameters:

t kre_thread structure with running task.

Returns:

Error output.

4.15 kre_utils.h File Reference

kREproxy utilities

```
#include <linux/kernel.h>
#include <linux/module.h>
#include "kre_debug.h"
```

Functions

- int kre_util_pow (int x, int y)
  Power x to y.
- int kre_util_chartoint (char *c)
  Convers char to integer.
- int kre_util_atoi (char *number, size_t s)
  Asci to int : table with number : size of number, can be 0 then strlen will be used.
- void kre_util_strreverse (char *rev, const char *str, size_t s)
  String reverse.

4.15.1 Detailed Description

kREproxy utilities

4.15.2 Function Documentation

4.15.2.1 int kre_util_chartoint (char * *c*)

Convers char to integer.

Parameters:

c Character to convers.

Returns:

Integer value of digit from c.

4.15.2.2 int kre_util_pow (int *x*, int *y*)

Power x to y.

Parameters:

x

y

Index