NAME

tcp_functions **-** Alternate TCP Stack Framework

SYNOPSIS

#include <netinet/tcp.h>
#include <netinet/tcp_var.h>

int
register_tcp_functions(*struct tcp_function_block *blk*, *int wait*);

int
register_tcp_functions_as_name(*struct tcp_function_block *blk*, *const char *name*, *int wait*);

register_tcp_functions_as_names(*struct tcp_function_block *blk*, *int wait*, *const char *names[]*,
*int *num_names*);

int
deregister_tcp_functions(*struct tcp_function_block *blk*);

DESCRIPTION

The **tcp_functions** framework allows a kernel developer to implement alternate TCP stacks. The alternate stacks can be compiled in the kernel or can be implemented in loadable kernel modules. This functionality is intended to encourage experimentation with the TCP stack and to allow alternate behaviors to be deployed for different TCP connections on a single system.

A system administrator can set a system default stack. By default, all TCP connections will use the system default stack. Additionally, users can specify a particular stack to use on a per-connection basis. (See tcp(4) for details on setting the system default stack, or selecting a specific stack for a given connection.)

This man page treats "TCP stacks" as synonymous with "function blocks". This is intentional. A "TCP stack" is a collection of functions that implement a set of behavior. Therefore, an alternate "function block" defines an alternate "TCP stack".

The **register_tcp_functions**(), **register_tcp_functions_as_name**(), and **register_tcp_functions_as_names**() functions request that the system add a specified function block and register it for use with a given name. Modules may register the same function block multiple times with different names. However, names must be globally unique among all registered function blocks. Also, modules may not ever modify the contents of the function block (including the name) after it has been registered, unless the module first successfully de-registers the function block.

The **register_tcp_functions**() function requests that the system register the function block with the name defined in the function block’s *tfb_tcp_block_name* field. Note that this is the only one of the three registration functions that automatically registers the function block using the name defined in the function block’s *tfb_tcp_block_name* field. If a module uses one of the other registration functions, it may request that the system register the function block using the name defined in the function block’s *tfb_tcp_block_name* field by explicitly providing that name.

The **register_tcp_functions_as_name**() function requests that the system register the function block with the name provided in the *name* argument.

The **register_tcp_functions_as_names**() function requests that the system register the function block with all the names provided in the *names* argument. The *num_names* argument provides a pointer to the number of names. This function will either succeed in registering all of the names in the array, or none of the names in the array. On failure, the *num_names* argument is updated with the index number of the entry in the *names* array which the system was processing when it encountered the error.

The **deregister_tcp_functions**() function requests that the system remove a specified function block from the system. If this call succeeds, it will completely deregister the function block, regardless of the number of names used to register the function block. If the call fails because sockets are still using the specified function block, the system will mark the function block as being in the process of being removed. This will prevent additional sockets from using the specified function block. However, it will not impact sockets that are already using the function block.

tcp_functions modules must call one or more of the registration functions during initialization and successfully call the **deregister_tcp_functions**() function prior to allowing the module to be unloaded.

The *blk* argument is a pointer to a *struct tcp_function_block*, which is explained below (see *Function Block Structure*). The *wait* argument is used as the *flags* argument to malloc(9), and must be set to one of the valid values defined in that man page.

Function Block Structure

The *blk argument is a pointer to a struct tcp_function_block*, which has the following members:

```
struct tcp_function_block {
          char      tfb_tcp_block_name[TCP_FUNCTION_NAME_LEN_MAX];
          int       (*tfb_tcp_output)(struct tcpcb *);
          void      (*tfb_tcp_do_segment)(struct mbuf *, struct tcphdr *,
                               struct socket *, struct tcpcb *,
                               int, int, uint8_t,
                               int);
```

```
              int     (*tfb_tcp_ctloutput)(struct socket *so,
                                  struct sockopt *sopt,
                                  struct inpcb *inp, struct tcpcb *tp);
              /* Optional memory allocation/free routine */
              void    (*tfb_tcp_fb_init)(struct tcpcb *);
              void    (*tfb_tcp_fb_fini)(struct tcpcb *, int);
              /* Optional timers, must define all if you define one */
              int     (*tfb_tcp_timer_stop_all)(struct tcpcb *);
              void    (*tfb_tcp_timer_activate)(struct tcpcb *,
                                  uint32_t, u_int);
              int     (*tfb_tcp_timer_active)(struct tcpcb *, uint32_t);
              void    (*tfb_tcp_timer_stop)(struct tcpcb *, uint32_t);
              /* Optional functions */
              void    (*tfb_tcp_rexmit_tmr)(struct tcpcb *);
              void    (*tfb_tcp_handoff_ok)(struct tcpcb *);
              /* System use */
              volatile uint32_t tfb_refcnt;
              uint32_t  tfb_flags;
      };
```

The *tfb_tcp_block_name* field identifies the unique name of the TCP stack, and should be no longer than TCP_FUNCTION_NAME_LEN_MAX-1 characters in length.

The *tfb_tcp_output*, *tfb_tcp_do_segment*, and *tfb_tcp_ctloutput* fields are pointers to functions that perform the equivalent actions as the default **tcp_output**(), **tcp_do_segment**(), and **tcp_default_ctloutput**() functions, respectively. Each of these function pointers must be non-NULL.

If a TCP stack needs to initialize data when a socket first selects the TCP stack (or, when the socket is first opened), it should set a non-NULL pointer in the *tfb_tcp_fb_init* field. Likewise, if a TCP stack needs to cleanup data when a socket stops using the TCP stack (or, when the socket is closed), it should set a non-NULL pointer in the *tfb_tcp_fb_fini* field.

If the *tfb_tcp_fb_fini* argument is non-NULL, the function to which it points is called when the kernel is destroying the TCP control block or when the socket is transitioning to use a different TCP stack. The function is called with arguments of the TCP control block and an integer flag. The flag will be zero if the socket is transitioning to use another TCP stack or one if the TCP control block is being destroyed.

If the TCP stack implements additional timers, the TCP stack should set a non-NULL pointer in the *tfb_tcp_timer_stop_all*, *tfb_tcp_timer_activate*, *tfb_tcp_timer_active*, and *tfb_tcp_timer_stop* fields. These fields should all be NULL or should all contain pointers to functions. The *tfb_tcp_timer_activate*, *tfb_tcp_timer_active*, and *tfb_tcp_timer_stop* functions will be called when the **tcp_timer_activate**(), **tcp_timer_active**(), and **tcp_timer_stop**() functions, respectively, are called with a timer type other than the standard types. The functions defined by the TCP stack have the same semantics (both for arguments and return values) as the normal timer functions they supplement.

Additionally, a stack may define its own actions to take when the retransmit timer fires by setting a non-NULL function pointer in the *tfb_tcp_rexmit_tmr* field. This function is called very early in the process of handling a retransmit timer. However, care must be taken to ensure the retransmit timer leaves the TCP control block in a valid state for the remainder of the retransmit timer logic.

A user may select a new TCP stack before calling connect(2) or listen(2). Optionally, a TCP stack may also allow a user to begin using the TCP stack for a connection that is in a later state by setting a non-NULL function pointer in the *tfb_tcp_handoff_ok* field. If this field is non-NULL and a user attempts to select that TCP stack after calling connect(2) or listen(2) for that socket, the kernel will call the function pointed to by the *tfb_tcp_handoff_ok* field. The function should return 0 if the user is allowed to switch the socket to use the TCP stack. Otherwise, the function should return an error code, which will be returned to the user. If the *tfb_tcp_handoff_ok* field is NULL and a user attempts to select the TCP stack after calling connect(2) or listen(2) for that socket, the operation will fail and the kernel will return EINVAL.

The *tfb_refcnt* and *tfb_flags* fields are used by the kernel's TCP code and will be initialized when the TCP stack is registered.

Requirements for Alternate TCP Stacks

If the TCP stack needs to store data beyond what is stored in the default TCP control block, the TCP stack can initialize its own per-connection storage. The *t_fb_ptr* field in the *struct tcpcb* control block structure has been reserved to hold a pointer to this per-connection storage. If the TCP stack uses this alternate storage, it should understand that the value of the *t_fb_ptr* pointer may not be initialized to NULL. Therefore, it should use a *tfb_tcp_fb_init* function to initialize this field. Additionally, it should use a *tfb_tcp_fb_fini* function to deallocate storage when the socket is closed.

It is understood that alternate TCP stacks may keep different sets of data. However, in order to ensure that data is available to both the user and the rest of the system in a standardized format, alternate TCP stacks must update all fields in the TCP control block to the greatest extent practical.

RETURN VALUES

The **register_tcp_functions**(), **register_tcp_functions_as_name**(), **register_tcp_functions_as_names**(), and **deregister_tcp_functions**() functions return zero on success and non-zero on failure. In particular, the **deregister_tcp_functions**() will return EBUSY until no more connections are using the specified TCP stack. A module calling **deregister_tcp_functions**() must be prepared to wait until all connections have stopped using the specified TCP stack.

ERRORS

The **register_tcp_functions**() function will fail if:

[EINVAL] Any of the members of the *blk* argument are set incorrectly.

[ENOMEM] The function could not allocate memory for its internal data.

[EALREADY] A function block is already registered with the same name.

The **deregister_tcp_functions**() function will fail if:

[EPERM] The *blk* argument references the kernel's compiled-in default function block.

[EBUSY] The function block is still in use by one or more sockets, or is defined as the current default function block.

[ENOENT] The *blk* argument references a function block that is not currently registered.

SEE ALSO

connect(2), listen(2), tcp(4), malloc(9)

HISTORY

This framework first appeared in FreeBSD 11.0.

AUTHORS

The **tcp_functions** framework was written by Randall Stewart <*rrs@FreeBSD.org*>.

This manual page was written by Jonathan Looney <*jtl@FreeBSD.org*>.