TN NET

TCP/IP stack sockets API

V. 0.8

(http://www.tnkernel.com/)

Document Disclaimer

The information in this document is subject to change without notice. While the information herein is assumed to be accurate, Yuri Tiomkin (the author) assumes no responsibility for any errors or omissions.
The author makes and you receive no warranties or conditions, express, implied, statutory or in any communications with you. The author specifically disclaims any implied warranty of merchantability or fitness for a particular purpose.

Copyright notice

Trademarks

Names mentioned in this manual may be trademarks of their respective companies.
Brand and product names are trademarks or registered trademarks of their respective holders.

Document Version:

- 0.8

Acknowledgments:

s_socket	Create an endpoint for communication

Function:

int s_socket (int *domain*,
int *type*,
int *protocol*)

Parameter:

domain	In TN NET, domain is always a **AF_INET**
type	**SOCK_DGRAM** – for UDP socket **SOCK_STREAM** – for TCP socket **SOCK_RAW** – for raw socket
protocol	**IPPROTO_UDP** – for SOCK_DGRAM type **0** – for all another types

Return parameter:

A socket descriptor	Normal completion
-EPERM (-1)	Operation failed

Description:

The *s_socket()* function creates an unbound socket, and returns a socket descriptor that can be used in later function calls that operate on sockets.

s_close	Causes the system to release resources allocated to a socket.

Function:

int s_close (int *s*)

Parameters:

s	Specifies the descriptor of the socket to be close

Return parameter:

0 (TERR_NO_ERR)	Normal completion
-EINVAL	Input parameter(s) has a wrong value
-EFAULT	Operation failed

Description:

This function causes the system to release resources allocated to a socket. In case of TCP socket, the connection is terminated.

s_bind	bind a name to a socket

Function:

int s_bind (int *s,*
***const struct _sockaddr ** name**,*
int** namelen**)

Parameter:

s	Specifies the descriptor of the socket to be bound
name	Points to a **_sockaddr** structure containing the address to be bound to the socket. The length and format of the address depend on the address family of the socket. In TN NET, a **_sockaddr** is actually a **sockaddr__in** structure
namelen	Specifies the length of the **sockaddr** structure pointed to by the *name* argument

Return parameter:

0 (TERR_NO_ERR)	Normal completion
-EINVAL	Input parameter(s) has a wrong value
-ENOBUFS	Out of memory
-EADDRNOTAVAIL	The specified address is not available from the local machine

Description:

The *s_bind()* function will assign a local socket address *name* to a socket identified by descriptor *s* that has no local socket address assigned. Sockets created with the *s_socket()* function are initially unnamed; they are identified only by their address family.

s_connect	Connect a socket

Function:

int s_connect (int *s,*
struct _sockaddr * *name***,**
int *namelen***)**

Parameter:

s	Specifies the descriptor of the socket to be connect
name	Points to a **_sockaddr** structure containing the peer address. The length and format of the address depend on the address family of the socket. In TN NET, **_sockaddr** is actually a **sockaddr__in** structure.
namelen	Specifies the length of the **_sockaddr** structure pointed to by the *name* argument.

Return parameter:

0 (TERR_NO_ERR)	Normal completion
-EINVAL	Input parameter(s) has a wrong value
-EPERM	Operation not permitted
-ENOBUFS	Out of memory
-EADDRNOTAVAIL	The specified address is not available from the local machine

Description:

The *s_connect*() function makes an attempt to set a connection on a socket.
If the socket has not already been bound to a local address, *s_connect*() will bind it to an address which is an unused local address.

If the socket is a SOCK_DGRAM or SOCK_RAW, a *s_connect*() function set the socket's peer address, and no connection is made. If *name* is a null address for the protocol, the socket's peer address will be reset.

If the socket is a SOCK_STREAM, then *s_connect*() will makes an attempt to establish a connection to the address specified by the *name* argument. If the connection cannot be established immediately, *s_connect*() will block for up to an unspecified timeout interval until the connection is established. If the timeout interval expires before the connection is established, *s_connect*() will fail and the connection attempt will be aborted.

s_accept	Accept a new connection on a socket

Function:

int s_accept (int *s*,
struct _sockaddr * *addr*,
int * *addrlen*)

Parameter:

s	Specifies a socket that was created with *s_socket()* has been bound to an address with *s_bind()* and has issued a successful call to *s_listen()*
addr	Either a null pointer, or a pointer to a **_sockaddr** structure where the address of the connecting socket will be returned. In TN NET, **_sockaddr** is actually a **sockaddr__in** structure.
addrlen	Points to a variable which on input specifies the length of the supplied **_sockaddr** structure, and on output specifies the length of the stored address.

Return parameter:

A descriptor of the accepted socket	Normal completion
-EPERM (-1)	Operation failed

Description:

This function works only with a SOCK_STREAM sockets.

The *s_accept*() function extracts the first connection on the queue of pending connections, creates a new socket with the same socket type protocol and address family as the specified socket, and allocates a new descriptor for that socket.

If *addr* is not a null pointer, the address of the peer for the accepted connection will be stored in the **_sockaddr** /**sockaddr__in** structure pointed to by *addr*, and the length of this address will be stored in the variable pointed to by *addrlen*.

If the listen queue is empty of connection requests *s_accept*() will block until a connection is present.

The accepted socket cannot itself accept more connections. The original socket remains open and can accept more connections.

s_listen	Listen for socket connections and limit the queue of incoming connections

Function:

int s_listen (int *s,*
int *backlog***)**

Parameter:

s	Specifies the descriptor of the socket
backlog	The maximum length of the queue of pending connections.

Return parameter:

0 (TERR_NO_ERR)	Normal completion
-EINVAL	Input parameter(s) has a wrong value
-EPERM	Operation not permitted

Description:

This function works only with a SOCK_STREAM sockets.

The *s_listen()* function marks a SOCK_STREAM socket, specified by the socket argument *s*, as accepting connections, and limits the number of outstanding connections in the socket's listen queue to the value specified by the *backlog* argument. The socket *s* is put into 'passive' mode where incoming connection requests are acknowledged and queued pending acceptance by the process.

s_recv	Receive a message from a connected socket

Function:

int s_recv (int *s*,
unsigned char * *buf*,
int *nbytes*,
int *flags*)

Parameter:

s	Specifies the socket descriptor.
buf	Points to a buffer where the message should be stored
nbytes	Specifies the length in bytes of the buffer pointed to by the *buf* argument.
flags	Specifies the type of message reception. IN TN NET, should be 0.

Return parameter:

The length of the message in bytes	Normal completion
-EINVAL	Input parameter(s) has a wrong value
-ENOTCONN	A receive is attempted on a connection-mode socket that is not connected
-ENOBUFS	Insufficient resources were available in the system to perform the operation.

Description:

This function works only with a SOCK_STREAM sockets in TN NET.
The *s_recv*() function returns the length of the message written to the buffer pointed to by the *buf* argument. For SOCK_STREAM sockets, message boundaries will be ignored. In this case, data will returned to the user as soon as it becomes available, and no data will be discarded.

If no messages are available at the socket and SS_NBIO flag is not set on the socket descriptor (default), *s_recv*() will block until a message arrives. If no messages are available at the socket and SS_NBIO flag is set on the socket's descriptor (by a *s_ioctl()* function), *s_recv*() will fail and returns a value (-EWOULDBLOCK).

In TN NET, in the most cases there is no reason to use non-blocking reception. When *s_recv*() will block, the task automatically switches to the WAIT state and gives an ability to run for another tasks.

s_recvfrom	Receive a message from a socket

Function:

```
int  s_recvfrom (int  s,
                 unsigned char *  buf,
                 int  len,
                 int  flags,
                struct _sockaddr * from,
                int  * fromlenaddr)
```

Parameter:

s	Specifies the socket descriptor
buf	Points to the buffer where the message should be stored.
len	Specifies the length in bytes of the buffer pointed to by the *buf* argument.
flags	Specifies the type of message reception. IN TN NET, should be 0.
from	A null pointer, or points to a **_sockaddr** structure in which the sending address is to be stored. In TN NET, **_sockaddr** is actually a **sockaddr__in** structure
fromlenaddr	Specifies the length of the **_sockaddr** structure pointed to by the *from* argument**.**

Return parameter:

The length of the message in bytes	Normal completion
-EINVAL	Input parameter(s) has a wrong value
-ENOTCONN	A socket that is not connected
-ENOBUFS	Insufficient resources to perform the operation.
-EWOULDBLOCK	No messages at the socket (non-blocking mode only)
-ETIMEOUT	Timeout occurred (blocking mode only)

Description:

This function works only with a SOCK_DGRAM and SOCK_RAW sockets in TN NET.

The *s_recvfrom*() function receives a message from a socket. It permits the application to retrieve the source address of received data.The *s_recvfrom*() function returns the length of the message written to the buffer pointed to by the *buf* argument. For SOCK_RAW and SOCK_DGRAM sockets, the entire will be read in a single operation.

If a message is too long to fit in the supplied buffer, the excess bytes will be discarded.

If the *from* argument is not a null pointer, the source address of the received message will be stored in the **_sockaddr** structure pointed to by the *from* argument, and the length of this address will be stored in the variable pointed to by the *fromlenaddr* argument.

If no messages are available at the socket and SS_NBIO flag is not set on the socket descriptor (default), *s_recvfrom*() will block until a message arrives. If no messages are available at the socket and SS_NBIO flag is set on the socket's descriptor (by a *s_ioctl()* function), *s_recvfrom*() will fail and returns a value (-EWOULDBLOCK).

In TN NET, in the most cases there is no reason to use non-blocking reception. When *s_recvfrom*() function will block, the task automatically switches to the WAIT state and gives an ability to run for another tasks. For UDP/RAW sockets, the timeout feature (see *s_setsockopt()* function) sets the timeout value that specifies the maximum amount of time an input function waits until it completes in the blocking mode.

s_send	Send a message on a socket

Function:

int s_send (int *s*,
unsigned char * *buf*,
int *nbytes*,
int *flags*)

Parameter:

s	Specifies the socket descriptor
buf	Points to the buffer containing the message to send.
nbytes	Specifies the length of the message in bytes.
flags	Specifies the type of message transmission. IN TN NET, should be 0.

Return parameter:

The number of bytes sent.	Normal completion
-EINVAL	Input parameter(s) has a wrong value
-ENOBUFS	Insufficient resources were available in the system to perform the operation.
-ENOTCONN	The socket is not connected or otherwise has not had the peer pre-specified.

Description:

This function works only with a SOCK_STREAM sockets in TN NET.
The *s_send*() function will send a message only when the socket is connected.
The length of the message to be sent is specified by the *nbytes* argument.
If the message is too long to pass through the underlying protocol, *s_send*() will fail and no data will be transmitted.
Successful completion of a call to *s_send*() does not guarantee delivery of the message. A return error value (if any) indicates only locally-detected errors.
If space is not available at the sending socket to hold the message to be transmitted, and SS_NBIO flag is not set on the socket descriptor (default), *s_send*() will block until space is available. If space is not available at the sending socket to hold the message to be transmitted, and SS_NBIO flag is set on the socket's descriptor (by a *s_ioctl()* function), *s_send*() will fail and returns a value (-EWOULDBLOCK).
In TN NET, in the most cases there is no reason to use non-blocking transmission. When *s_send*() will block, the task automatically switches to the WAIT state and gives an ability to run for another tasks.

s_sendto Send a message on a socket

Function:

```
int  s_sendto (int s,
               unsigned char * buf,
               int len,
               int flags,
               struct _sockaddr * dst_addr,
               int len)
```

Parameter:

s	Specifies the socket descriptor
buf	Points to a buffer containing the message to be sent.
len	Specifies the size of the message in bytes.
flags	Specifies the type of message transmission. IN TN NET, should be 0.
dst_addr	NULL or points to a **_sockaddr** structure containing the destination address. In TN NET, **_sockaddr** is actually a **sockaddr__in** structure
len	Specifies the length of the **_sockaddr** structure pointed to by the *dst_addr* argument.

Return parameter:

The number of bytes sent	Normal completion
-EINVAL	Input parameter(s) has a wrong value
-ENOBUFS	Insufficient resources were available in the system to perform the operation.
-ENOTCONN	The socket is not connected or otherwise has not had the peer pre-specified.

Description:

This function works only with a SOCK_DGRAM and SOCK_RAW sockets in TN NET.

The *s_sendto*() function will send a message through a socket. The message will be sent to the address specified by *dst_addr* argument. The *len* argument specifies the length of the message.

In TN NET if a *dst_addr* argument is NULL, a destination address should be set before *s_sendto*() function using - by the function's *s_connect()* call.

Successful completion of a call to *s_sendto*() does not guarantee delivery of the message. A return value of error (if any) indicates only locally-detected errors.

If space is not available at the sending socket to hold the message to be transmitted, and SS_NBIO flag is not set on the socket descriptor (default), *s_sendto*() will block until space is available. If space is not available at the sending socket to hold the message to be transmitted, and SS_NBIO flag is set on the socket's descriptor (by a *s_ioctl()* function), *s_sendto*() will fail and returns a value (-EWOULDBLOCK).

In TN NET, in the most cases there is no reason to use non-blocking transmission. When *s_sendto*() will block, the task automatically switches to the WAIT state and gives an ability to run for another tasks.

s_shutdown Shut down socket send and receive operations

Function:

int s_shutdown (int *s*,
int *how***)**

Parameter:

s	Specifies the socket descriptor	
how	Specifies the type of shutdown. The values are as follows:	
	SHUT_RD	- Disables further receive operations
	SHUT_WR	- Disables further send operations.
	SHUT_RDWR	- Disables further send and receive operations.

Return parameter:

0 (TERR_NO_ERR)	Normal completion
-EINVAL	Input parameter(s) has a wrong value
-ENOTCONN	The socket is not connected.

Description:

This function works only with a SOCK_STREAM sockets in TN NET.

The *s_shutdown*() function will cause all or part of a full-duplex connection on the socket associated with the descriptor *s* to be shut down.

The *s_shutdown*() function disables subsequent send and/or receive operations on a socket, depending on the value of the *how* argument.

s_ioctl	Control a socket behavior

Function:

int s_ioctl (int * *s*,
int *cmd*,
void * *data*)

Parameters:

s	Specifies the socket descriptor
cmd	Specifies the command to the socket *s*
data	Points to the memory in which the *cmd* data is stored to put into the socket or will be obtained from the socket after *cmd* executing

Return parameter:

0 (TERR_NO_ERR)	Normal completion
-EINVAL	Input parameter(s) has a wrong value
-ENOPROTOOPT	Unknown / unsupported *cmd*

Description:

In TN NET, for a SOCK_STREAM sockets, a parameter *cmd* can has a value:

- **_FIONBIO** - set/remove the socket's SS_NBIO flag

  It is a socket source code fragment for the **_FIONBIO** cmd:

```
if(*((int *)data))
   so->so_state |= SS_NBIO;
else
   so->so_state &= ~SS_NBIO;
return 0;
```

- **_FIONREAD** - get actual bytes number in the socket's reception buffer

  It is a socket source code fragment for the **_FIONREAD** cmd:

```
*((int *)data) = so->so_rcv.sb_cc;
return (0);
```

In TN NET, for a SOCK_DGRAM and SOCK_RAW sockets, a parameter *cmd* can has a value:

- **_FIONBIO** - set/remove the socket's SS_NBIO flag (the same as for a SOCK_STREAM sockets)
- **_FIORXTIMEOUT -** set the socket reception timeout value (in milliseconds)

  It is a socket source code fragment for the **_FIORXTIMEOUT** cmd:

```
unsigned int rc;
bcopy(data,(unsigned char*)&rc, sizeof(int));
so->rx_timeout = rc;
return 0; //-- O.K
```

s_getpeername Get the name of the peer socket

Function:

int s_getpeername (int *s*,
struct _sockaddr * *name*,
int * *namelen*)

Parameters:

s	Specifies the socket descriptor
name	Points to a **_sockaddr** structure to store the peer socket name. In TN NET, **_sockaddr** is actually a **sockaddr__in** structure.
namelen	The length of *name* argument in the variable pointed to by the *namelen* argument.

Return parameter:

0 (TERR_NO_ERR)	Normal completion
-EINVAL	Input parameter(s) has a wrong value
-ENOTCONN	A receive is attempted on a connection-mode socket that is not connected
-ENOBUFS	Insufficient resources were available in the system to perform the operation.

Description:

The s_g*etpeername*() function retrieves the peer address of the specified socket, stores this address in the **_sockaddr** structure pointed to by the *name* argument, and stores the length of this address in the variable pointed to by the *namelen* argument.

If the actual length of the address is greater than the length of the supplied **_sockaddr** structure, the stored address will be truncated.

If the protocol permits connections by unbound clients, and the peer is not bound, then the value stored in the object pointed to by *name* is unspecified.

s_setsockopt Set the socket options

Function:

int s_setsockopt (int *s*,
int *level*,
int *name*,
unsigned char * *val*,
int *avalsize*)

Parameters:

s	Specifies the socket descriptor
level	In TN NET, should be 0
name	The *name* argument specifies a single option to set
val	Pointed to by the memory buffer that contains a single option data to set
avalsize	Size of the option data

Return parameter:

0 (TERR_NO_ERR)	Normal completion
-EINVAL	Input parameter(s) has a wrong value
-ENOPROTOOPT	The option is not supported

Description:

The supported options are as follows:

SO_LINGER (TCP only)

Lingers on a *s_close()* data is present. This option controls the action taken when unsent messages queue on a socket and *s_close()* is performed. If SO_LINGER is set, the system will block the calling thread during *s_close()* until it can transmit the data or until the time expires. If SO_LINGER is not specified, and *s_close()* is issued, the system handles the call in a way that allows the calling thread to continue as quickly as possible. This option takes a **__linger** structure, as defined in the "*bsd_socket.h*" header, to specify the state of the option and linger interval.

SO_KEEPALIVE (TCP only)

Keeps connections active by enabling the periodic transmission of messages.This option takes an **int** value. This is a Boolean option.

SO_SNDBUF (TCP only)

Sets send buffer size. This option takes an **int** value.

SO_RCVBUF (TCP only)

Sets receive buffer size. This option takes an **int** value.

SO_RCVLOWAT (TCP only)

Sets the minimum number of bytes to process for socket input operations. The default value for SO_RCVLOWAT is 1. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. (They may return less than the low water mark if an error occurs or the type of data next in the receive queue is different from that returned; for example, out-of-band data.) This option takes an **int** value.

SO_RCVTIMEO (UDP/RAW only)

Sets the timeout value that specifies the maximum amount of time an input function waits until it completes. This option takes an **int** value. The timeout value is in milliseconds.
The default for this option is TN_WAIT_INFINITE, which indicates that a receive operation will not time out.

SO_SNDLOWAT (TCP only)

Sets the minimum number of bytes to process for socket output operations. Non-blocking output operations will process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. This option takes an **int** value.

For Boolean options, **0** indicates that the option is disabled and **1** indicates that the option is enabled.

s_getsockopt	Get the socket options

Function:

```
int  s_getsockopt (int s,
                   int level,
                   int name,
                   unsigned char * val,
                   int avalsize)
```

Parameters:

s	Specifies the socket descriptor
level	In TN NET, should be 0
name	The *name* argument specifies a single option to get
val	Pointed to by the memory buffer that will contains a single option data
avalsize	Size of the option data

Return parameter:

0 (TERR_NO_ERR)	Normal completion
-EINVAL	Input parameter(s) has a wrong value
-ENOPROTOOPT	The option is not supported

Description:

This function is a “mirror” functions to the *s_setsockopt()* function and lets to obtain an option data/value. A supported options are the same as for the *s_setsockopt()* function.

Appendix A:

This is a **_sockaddr** and **sockaddr__in** structures definition:

```
/*
 *  There are a BSD 'in_addr', 's_addr', ‘sockaddr’, ‘sockaddr__in’
 *  structures, extra '_' characther was added to resolve
 *  a Win32 names conflict only
 */

/* packed */

struct in__addr
{
   unsigned int s__addr;
};

/* packed */

struct _sockaddr
{
   unsigned char sa_len;          /* total length */
   unsigned char sa_family;       /* address family */
   char sa_data[14];              /* actually longer; address value */
};


/* packed */

struct sockaddr__in       /* Socket address, internet style */
{
   unsigned char sin_len;
   unsigned char sin_family;
   unsigned short sin_port;
   struct  in__addr sin_addr;
   char sin_zero[8];
};
```