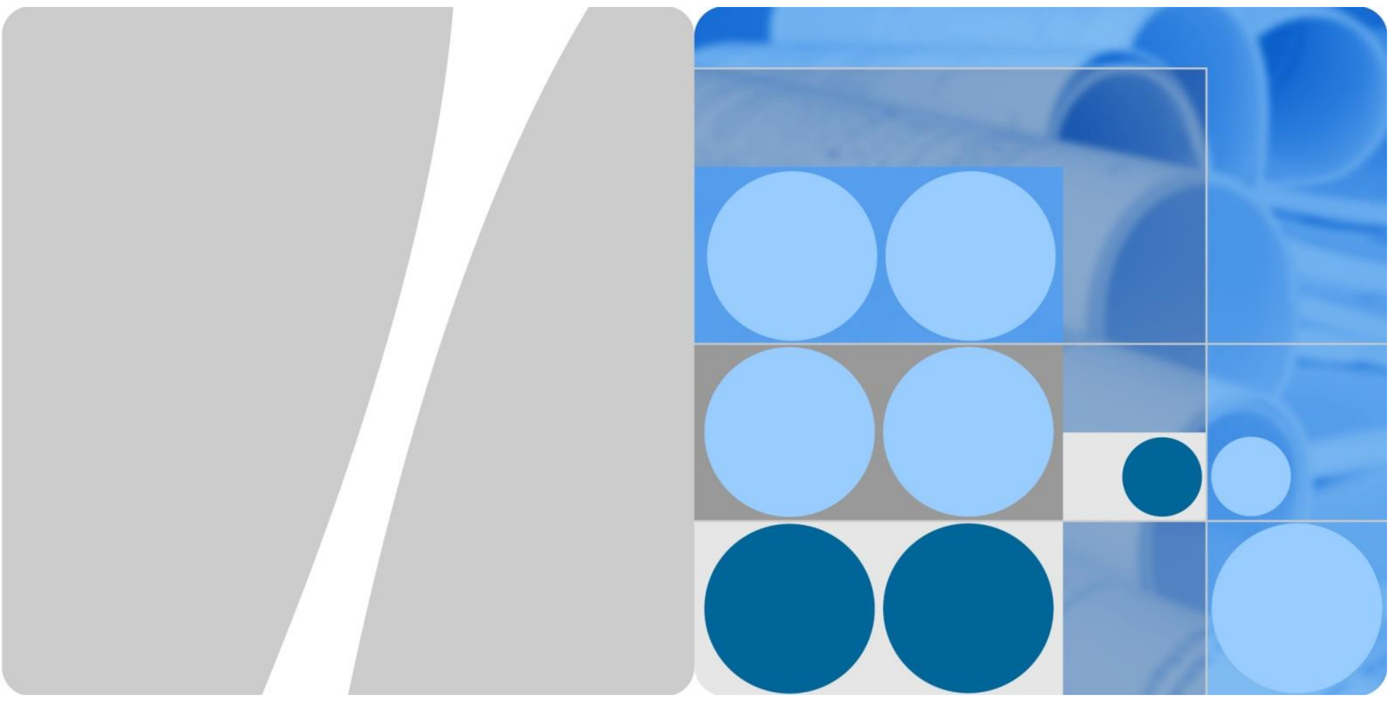

V100R001C20 SPC006 Huawei LiteOS lwIP API Reference

Date: **2016-11-22**

HUAWEI TECHNOLOGIES CO., LTD.

VPP2.0 V100R001C20 Huawei LiteOS lwIP API Reference API Reference

Here is a list of all interfaces and methods:

Driver_Interfaces

- **Driverif_input**

Detailed Description

This section contains the Network Driver related interfaces.

Driverif_input

Prototype

```
void  driverif_input(struct netif *netif, struct pbuf *p);
```

Purpose

This function should be called by network driver to pass the input packet to lwIP.

Description

This function should be called by network driver to pass the input packet to lwIP. Before calling this API, driver has to keep the packet in pbuf structure. Driver has to call pbuf_alloc() with type as PBUF_RAM to create pbuf structure. Then driver has to pass the pbuf structure to this API. This will add the pbuf into the TCPIP thread. Once this packet is processed by TCPIP thread, pbuf will be freed. Driver is not required to free the pbuf.

Parameters:

netif The lwIP network interface [N/A]

p packet in pbuf structure [N/A]

Return values

None

Required Header File

driverif.h

Note

N/A

Related Topics

N/A

Ifaddrs

```
  struct ifaddrs {

 struct ifaddrs  *ifa_next;

 char    *ifa_name;

 unsigned int   ifa_flags;

 struct sockaddr *ifa_addr;

 struct sockaddr *ifa_netmask;

 struct sockaddr *ifa_dstaddr;

 void    *ifa_data;

};
```

:

ifa_next Pointer to the next network interfaces.

ifa_name Pointer to the name of the network interfaces

ifa_flags Pointer to the status flag.

ifa_addr Pointer to the address.

ifa_netmask Pointer to the net masking.

ifa_dstaddr Pointer to the destination address.

ifa_data Pointer to the data.

Network Interfaces Info

- **Network Interfaces Datastructure**
- **Getifaddrs**
- **Freeifaddrs**
- **Netif_set_status_callback**
- **Netif_set_remove_callback**

Detailed Description

This section contains the interfaces to get information about Network interfaces in lwIP.

Network Interfaces Datastructure

- **Ifaddrs**

Getifaddrs

Prototype

```
int getifaddrs(struct ifaddrs **ifap);
```

Purpose

This function creates a linked list of structures describing the network interfaces of the local system, and stores the address of the first item of the list in *ifap.

Description

This function creates a linked list of struct ifaddrs, which holds the information about the network interfaces of the local system. The ifa_next field contains a pointer to the next structure on the list, or NULL if this is the last item of the list. The ifa_name points to the null-terminated interface name. The ifa_flags field contains the interface flags. The ifa_addr field points to a structure containing the interface address. The ifa_netmask field points to a structure containing the netmask associated with ifa_addr, if applicable for the address family. Depending on whether the bit IFF_brcast or IFF_POINTOPOINT is set in ifa_flags (only one can be set at a time), either ifa_broadaddr will contain the brcast address associated with ifa_addr (if applicable for the address family) or ifa_dstaddr will contain the destination address of the point-to-point interface.

Parameters:

ifap Double Pointer to struct ifaddrs. [N/A]

Return values

ERR_OK : on success Non zero value : on failure

Required Header File

ifaddrs.h

Note

The data returned by getifaddrs() is dynamically allocated and should be freed using freeifaddrs() when no longer needed.

Related Topics

N/A

Freeifaddrs

Prototype

```
void freeifaddrs(struct ifaddrs *ifa);
```

Purpose

This function is to free the memory of struct ifaddrs * provided by getifaddrs.

Description

The function getifaddrs, provides the list of network interfaces in struct ifaddrs*. The application has to free the memory of struct ifaddrs * by using this function.

Parameters:

ifa Pointer to struct ifaddrs [N/A]

Return values

None

Required Header File

ifaddrs.h

Note

N/A

Related Topics

N/A

In_addr

Prototype

```
struct in_addr

{

 u32_t s_addr;

};
```

Description

Provides a binary notation of IPv4 address.

:

s_addr Specifies the s_addr.

INET_Interfaces

- **INET_Structures**
- **Inet_addr**
- **Inet_aton**
- **Inet_ntoa**
- **Inet_ntoa_r**

Detailed Description

This section contains the INET interfaces.

INET_Structures

- **In_addr**

Detailed Description

This section contains the INET data structures.

Inet_addr

Prototype

```
unsigned int  inet_addr (const char * cp);
```

Purpose

This function is used to convert the IPv4 address from string notation (number and dot format) to binary notation of network byte order.

Description

This function is used to convert the IPv4 address from string notation (number and dot format) to binary notation of network byte order.

Parameters:

cp Pointer to IPv4 address string [N/A]

Return values

Valid IP_add: On success, in unsigned int (32bit) format

IPADDR_NONE : On failure

Required Header File

inet.h

Note

1. This interface will be available as function if LWIP_INET_ADDR_FUNC is enabled, otherwise it will be available as macro.

Related Topics

N/A

Inet_aton

Prototype

```
int inet_aton(const char *cp, struct in_addr *addr)
```

Purpose

This function is used to convert the IPv4 address from string notation (number and dot format) to binary notation of network byte order and stores it in the structure that addr points to.

Description

This function is used to convert the IPv4 address from string notation (number and dot format) to binary notation of network byte order and stores it in the structure that addr points to.

Parameters:

cp Pointer to IPv4 address string [N/A]

addr Generated binary notation of IPv4 address will be udpated here [N/A]

Return values

1 : On success

0 : On failure

Required Header File

inet.h

Note

1. This interface is same as inet_addr(). Only difference is that the generated binary notation of IPv4 address is updated in the input parameter "addr", instead of returning it. 2. This interface is available as a function if LWIP_INET_ATON_FUNC is enabled, otherwise it is available as a macro.

Related Topics

N/A

Inet_ntoa

Prototype

```
char *inet_ntoa (struct in_addr in);
```

Purpose

This function is used to convert the IPv4 address from binary notation (network byte order) to string notation (number and dot format). This API is not reentrant safe.

Description

This function is used to convert the IPv4 address from binary notation (network byte order) to string notation (number and dot format). This API is not reentrant safe.

Parameters:

in Pointer to binary notation of IPv4 address [N/A]

Return values

Valid pointer : On success, returns pointer to string notation of IPv4 address

NULL : On failure

Required Header File

inet.h

Note

1. This interface will be available as function if LWIP_INET_NTOA_FUNC is enabled, otherwise it will be available as macro.

Related Topics

N/A

Inet_ntoa_r

Prototype

```
char *inet_ntoa_r(const ip_addr_t addr, char *buf, int buflen)
```

Purpose

This function is used to convert the IPv4 address from binary notation (network byte order) to string notation (number and dot format). This is the reentrant API of inet_ntoa.

Description

This function is used to convert the IPv4 address from binary notation (network byte order) to string notation (number and dot format). This is the reentrant API of inet_ntoa. So the generated string notation IPv4 address is updated in the user passed buffer.

Parameters:

addr Binary notation of IPv4 address [N/A]

buf Pointer to user buffer, in which the ouput string format of IPv4 address will be updated [N/A]

buflen Length of the user buffer [N/A]

Return values

Valid pointer : On success, returns pointer to buffer passed by user in buf

NULL : On failure

Required Header File

inet.h

Note

1. This interface is available as only macro.

Related Topics

N/A

Threadsafe_Network_Interfaces

- **Netifapi_netif_add**
- **Netifapi_netif_second_if_add**
- **Netifapi_netif_set_addr**
- **Netifapi_netif_common**
- **Netifapi_netif_remove**
- **Netifapi_netif_set_up**
- **Netifapi_netif_set_down**
- **Netifapi_netif_set_default**
- **Netifapi_netif_set_link_up**
- **Netifapi_netif_set_link_down**
- **Netifapi_autoip_start**
- **Netifapi_autoip_stop**
- **Netifapi_netif_set_link_callback**
- **Netifapi_netif_set_mtu**
- **Netifapi_stop_queue**
- **Netifapi_wake_queue**
- **Netif_status_callback_fn**
- **Netif**

Detailed Description

This section contains the Thread safe Network related interfaces.

Threadsafe_DHCP_Interfaces

- **Netifapi_dhcp_is_bound**
- **Netifapi_dhcp_start**
- **Netifapi_dhcp_stop**
- **Netifapi_dhcp_cleanup**
- **Netifapi_dhcp_inform**
- **Netifapi_dhcp_set_struct**
- **Netifapi_dhcp_remove_struct**
- **Netifapi_dhcps_start**
- **Netifapi_dhcps_stop**

Detailed Description

This section contains the Thread safe DHCP interfaces.

Netifapi_netif_add

Prototype

```
 err_t netifapi_netif_add(struct netif *netif, ip_addr_t *ipaddr, ip_addr_t *netmask,
ip_addr_t *gw);
```

Purpose

This API is used to add a network interface to the list of lwIP netifs.

Description

This is a thread safe API, used to add a network interface to the list of lwIP netifs. It is recommended to use this API instead of netif_add()

Parameters:

netif pre-allocated netif structure [N/A]

ipaddr IP_add for the new netif [N/A]

netmask network mask for the new netif [N/A]

gw default gateway IP_add for the new netif [N/A]

Return values

0 : On success

Negative value : On failure

Required Header File

netifapi.h

Note

1. The interface names are changed to the format <ifname><num>:<interface id>=""> For example: eth0:0. The interface id 0 is assigned to the main interface and 1 is assigned to the secondary interface

Related Topics

netif_add

Netifapi_netif_second_if_add

Prototype

```
 err_t netifapi_netif_second_if_add(struct netif *main_netif, struct netif *netif,
ip_addr_t *ipaddr, ip_addr_t *netmask, ip_addr_t *gw)
```

Purpose

This API is used to add a second IP to the already existing interface. The secondary interface is always static

Description

This API is used to add a second IP to the already existing interface. The secondary interface is always static

Parameters:

main_netif The lwIP network interface for which a secondary IP is to be added [N/A]

netif The secondary lwIP network interface [N/A]

ipaddr IP_add for the new netif [N/A]

netmask network mask for the new netif [N/A]

gw default gateway IP_add for the new netif [N/A]

Return values

0 : On success

Negative value : On failure

Required Header File

netifapi.h

Note

1. The interface names are changed to the format <ifname><num>:<interface id>=""> For example: eth0:0. The interface id 0 is assigned to the main interface and 1 is assigned to the

secondary interface. So eth0:0 indicates main ethernet interface and eth0:1 indicates secondary interface linked with eth0

Related Topics

netif_add

Netifapi_netif_set_addr

Prototype

```
 err_t netifapi_netif_set_addr(struct netif *netif, ip_addr_t *ipaddr, ip_addr_t *netmask,
ip_addr_t *gw );
```

Purpose

This API is used to change IP_add configuration for a network interface (including netmask and default gateway).

Description

This is a thread safe API, used to change IP_add configuration for a network interface (including netmask and default gateway). It is recommended to use this API instead of netif_set_addr()

Parameters:

netif network interface to be changed [N/A]

ipaddr new IP_add [N/A]

netmask new network mask [N/A]

gw new default gateway IP_add [N/A]

Return values

0 : On success

Negative value : On failure

Required Header File

netifapi.h

Note

N/A

Related Topics

netif_set_addr

Netifapi_netif_common

Prototype

```
 err_t netifapi_netif_common(struct netif *netif, netifapi_void_fn voidfunc,
netifapi_errt_fn errtfunc);
```

Purpose

This API is used to call all netif related APIs in a thread safe manner.

Description

This API is used to call all netif related APIs in a thread safe manner. Those netif related APIs should be of prototype to receive only struct netif* as argument and return type can of type err_t or void. User should pass either viodfunc or errtfunc.

Parameters:

netif Indicates the network interface to be passed as argument. [N/A]

voidfunc Indicates the netif API to call. [N/A]

errtfunc Indicates the netif API to call. [N/A]

Return values

0 : On success

Negative value : On failure

Required Header File

netifapi.h

Note

The prototype for netifapi_void_fn and netifapi_errt_fn are as follows:

typedef void (*netifapi_void_fn)(struct netif *netif);

typedef err_t (*netifapi_errt_fn)(struct netif *netif);

Related Topics

N/A

Netifapi_netif_remove

Prototype

```
err_t netifapi_netif_remove(struct netif * netif);
```

Purpose

This API is used to remove a network interface from the list of lwIP netifs in a thread-safe way.

Description

The etherent driver call this API in a thread-safe way to remove a network interface from the list of lwIP netifs.

Parameters:

netif network interface to be removed [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments

Required Header File

netifapi.h

Note

- Application send data will return success, but not send the data if only netifapi_netif_remove() API is called and not the sockets.
- Adaptor must close all the connection on socket before netifapi_netif_remove() API is called as netifapi_netif_remove() does not close established connections.

Related Topics

N/A

Netifapi_netif_set_up

Prototype

```
err_t netifapi_netif_set_up(struct netif *netif);
```

Purpose

This API is used to bring an interface up in a thread-safe way.

Description

This API is used to bring an interface up in a thread-safe way. i.e. available for processing traffic.

Parameters:

netif network interface [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments

Required Header File

netifapi.h

Note

Enabling DHCP on a down interface will make it come up once configured.

Related Topics

netifapi_dhcp_start

Netifapi_netif_set_down

Prototype

```
err_t netifapi_netif_set_down(struct netif *netif);
```

Purpose

This API is used to bring an interface down in a thread-safe way.

Description

This API is used to bring an interface up in a thread-safe way. i.e. disabling any traffic processing.

Parameters:

netif network interface [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments

Required Header File

netifapi.h

Note

Enabling DHCP on a down interface will make it come up once configured.

Related Topics

netifapi_dhcp_start

Netifapi_netif_set_default

Prototype

```
err_t netifapi_netif_set_default(struct netif *netif);
```

Purpose

This API is used to set a network interface as the default network interface in a thread-safe way.

Description

This API is used to set a network interface as the default network interface. It is used to output all packets for which no specific route is found.

Parameters:

netif network interface to be set as default [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments

Required Header File

netifapi.h

Note

N/A

Related Topics

N/A

Netifapi_netif_get_default

Prototype

```
struct netif *netifapi_netif_get_default(void);
```

Purpose

This API is used to get the default network interface in a thread-safe way.

Description

This API is used to get the default network interface. It is used to output all packets for which no specific route is found.

Parameters:

N/A

Return values

NULL : if the default netif was NOT exist or the default netif was down \n

Netif: the default network interface

Required Header File

netifapi.h

Note

N/A

Related Topics

N/A

Netifapi_netif_set_link_up

Prototype

```
err_t netifapi_netif_set_link_up(struct netif *netif);
```

Purpose

This thread-safe API is called by the driver when its link goes up.

Description

This thread-safe API is called by the driver when its link goes up.

Parameters:

netif network interface [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments

Required Header File

netifapi.h

Note

N/A

Related Topics

N/A

Netifapi_netif_set_link_down

Prototype

```
err_t netifapi_netif_set_link_down(struct netif *netif);
```

Purpose

This thread-safe API is called by the driver when its link goes down.

Description

This thread-safe API is called by the driver when its link goes down.

Parameters:

netif network interface [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments

Required Header File

netifapi.h

Note

N/A

Related Topics

N/A

Netifapi_set_hostname

Prototype

```
err_t netifapi_set_hostname(struct netif *netif, char *hostname, u8_t namelen);
```

Purpose

This API is used to set the hostname of the netif.

Description

This API is used to set the hostname of the netif, which is using in DHCP message. The hostname string length should be less than NETIF_HOSTNAME_MAX_LEN, otherwise the hostname will truncate to (NETIF_HOSTNAME_MAX_LEN-1).

Parameters:

netif Indicates the lwIP network interface. [N/A]

hostname The new hostname to use. [N/A]

namelen The hostname string length, should be within the scope of 0~255. [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments

Required Header File

netifapi.h

Note

N/A

Related Topics

N/A

Netifapi_get_hostname

Prototype

```
err_t netifapi_get_hostname(struct netif *netif, char *hostname, u8_t namelen);
```

Purpose

This API is used to get the hostname of the netif.

Description

This API is used to get the hostname of the netif, which is using in DHCP message. The hostname buffer length shoud not smaller than NETIF_HOSTNAME_MAX_LEN, otherwise it will get a truncated hostname.

Parameters:

netif Indicates the lwIP network interface. [N/A]

hostname The buffer to stroe hostname string of the netif. [N/A]

namelen The hostname string buffer length. [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments

Required Header File

netifapi.h

Note

N/A

Related Topics

N/A

Netifapi_dhcp_is_bound

Prototype

```
err_t netifapi_dhcp_is_bound(struct netif *netif);
```

Purpose

This API is used to get DHCP negotiation status for a network interface in a thread-safe way.

Description

This API is used to get DHCP negotiation status for a network interface in a thread-safe way.

Parameters:

netif The lwIP network interface [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments

ERR_INPROGRESS: On failure due to dhcp bind in progress.

Required Header File

netifapi.h

Note

N/A

Related Topics

N/A

Netifapi_dhcp_start

Prototype

```
err_t netifapi_dhcp_start(struct netif *netif);
```

Purpose

This API is used to start DHCP negotiation for a network interface in a thread-safe way.

Description

This API is used to start DHCP negotiation for a network interface. If no DHCP client instance is attached to this interface, a new client is created first. If a DHCP client instance is already present, it restarts negotiation. it is the thread-safe way for calling dhcp_start in user space.

Parameters:

netif The lwIP network interface. [N/A]

Return values

ERR_OK: On success

ERR_MEM: On failure

ERR_ARG:Invalid input parameter

Required Header File

netifapi.h

Note

N/A

Related Topics

N/A

Netifapi_dhcp_stop

Prototype

```
err_t netifapi_dhcp_stop(struct netif *netif);
```

Purpose

This API is used to remove the DHCP client from the interface in a thread-safe way.

Description

This API is used to remove the DHCP client from the interface. It stops DHCP configuration. It is the thread-safe way for calling dhcp_stop in user space.

Parameters:

netif The lwIP network interface [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments

Required Header File

netifapi.h

Note

N/A

Related Topics

N/A

Netifapi_dhcp_cleanup

Prototype

```
err_t netifapi_dhcp_cleanup(struct netif *netif);
```

Purpose

This API is used to free the memory allocated for dhcp during dhcp start.

Description

This API is used to free the memory allocated for dhcp during dhcp start. It is the thread-safe way for calling dhcp_cleanup in user space.

Parameters:

netif The lwIP network interface [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments

Required Header File

netifapi.h

Note

N/A

Related Topics

N/A

Netifapi_dhcp_inform

Prototype

```
err_t netifapi_dhcp_inform(struct netif *netif);
```

Purpose

This API is used to inform a DHCP server of our manual configuration.

Description

This API is used to inform a DHCP server of our manual configuration. This informs DHCP servers of our fixed IP_add configuration by sending an INFORM message. It does not involve DHCP address configuration, it is just here to be nice to the network.

Parameters:

netif lwIP network interface [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments

Required Header File

netifapi.h

Note

N/A

Related Topics

N/A

Netifapi_dhcp_set_struct

Prototype

```
err_t netifapi_dhcp_set_struct(struct netif *netif, struct dhcp *dhcp);
```

Purpose

This API is used to set a static dhcp structure to the netif.

Description

This API is used to set a static dhcp structure to the netif.

Parameters:

netif The lwIP network interface [N/A]

dhcp The dhcp struct which needs to set [N/A]

Return values

ERR_OK: On success

Required Header File

netifapi.h

Note

1. If this is used before netifapi_dhcp_start, then application needs to use netifapi_dhcp_remove_struct instead of netifapi_dhcp_cleanup to remove the struct dhcp.

Related Topics

netifapi_dhcp_remove_struct

Netifapi_dhcp_remove_struct

Prototype

```
err_t netifapi_dhcp_remove_struct(struct netif *netif);
```

Purpose

This API is used to remove the static dhcp structure from netif.

Description

This API is used to remove the static dhcp structure from netif, which was set using the API netifapi_dhcp_set_struct().

Parameters:

netif The lwIP network interface [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments

Required Header File

netifapi.h

Note

1. Application needs to use this API instead of netifapi_dhcp_cleanup, if the dhcp structure is previously set on netif using netifapi_dhcp_set_struct().

Related Topics

netifapi_dhcp_set_struct

Netifapi_dhcps_start

Prototype

```
err_t netifapi_dhcps_start(struct netif *netif, const char *start_ip, u16_t ip_num);
```

Purpose

This API is used to start DHCP Server negotiation for a network interface.

Description

This API is used to start DHCP Server negotiation for a network interface. If no DHCP serverwas attached to this interface, a new server is created first. It is the thread-safe way for calling dhcps_start in the user space. If start_ip is set(not NULL) and ip_num is bigger than 0, then dhcp server will use it as the start dhcp lease of dhcp server. the start_ip should in the same subnet of the netif. ip_num shoud not bigger than LWIP_DHCPS_MAX_LEASE, and make sure the last dhcp lease also in the same subnet of the netif. If start_ip is not set, or ip_num is 0, the dhcp lease will set start ip lease at subnet.1 and end at last valid ip addreess of the subnet(the ip lease number still limited by LWIP_DHCPS_MAX_LEASE).

Parameters:

netif The LwIP network interface. [N/A]

start_ip The start of ip address of dhcp lease. [N/A]

ip_num The size the dhcp lease pool. [N/A]

Return values

ERR_OK: On success

ERR_MEM: On failure.

ERR_ARG: Invalid input parameter.

Required Header File

netifapi.h

Note

N/A

Related Topics

N/A

Netifapi_dhcps_stop

Prototype

```
err_t netifapi_dhcps_stop(struct netif *netif);
```

Purpose

This API is used to remove the DHCP Server from the interface.

Description

This API is used to remove the DHCP Server from the interface. It stops DHCP Server configuration. It is the thread-safe way for calling dhcps_start in user space.

Parameters:

netif The lwIP network interface [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments

Required Header File

netifapi.h

Note

N/A

Related Topics

N/A

Netifapi_autoip_start

Prototype

```
err_t netifapi_autoip_start(struct netif *netif);
```

Purpose

Thread-safe API to start the AutoIP client.

Description

Thread-safe API to start the AutoIP client.

Parameters:

netif The lwIP network interface. [N/A]

Return values

ERR_OK: On success

ERR_MEM: On failure due to memory

ERR_VAL: On failure due to Illegal value

Required Header File

netifapi.h

Related Topics

N/A

Netifapi_autoip_stop

Prototype

```
err_t netifapi_autoip_stop(struct netif *netif);
```

Purpose

Thread-safe API to stop the AutoIP client.

Description

Thread-safe API to stop the AutoIP client.

Parameters:

netif Use only for functions where there is only "netif" parameter. [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments

Required Header File

netifapi.h

Note

Need to call this autoip_stop API to stop the AUTOIP service.

Related Topics

N/A

Netifapi_netif_set_link_callback

Prototype

```
 err_t netifapi_netif_set_link_callback(struct netif *netif, netif_status_callback_fn
link_callback);
```

Purpose

This API is used to set callback to netif. This will be called whenever link is brought up /down.

Description

This API is used to set callback to netif. This will be called whenever link is brought up /down.

Parameters:

netif The lwIP network interface. [N/A]

link_callback Function prototype for netif status- or link-callback functions. [N/A]

Return values

ERR_OK: This API always returns this value.

Required Header File

netifapi.h

Note

N/A

Related Topics

N/A

Netifapi_netif_set_mtu

Prototype

```
err_t netifapi_netif_set_mtu(struct netif *netif, u16_t mtu);
```

Purpose

This API is used to set mtu of the netif. This will be called whenever mtu needs to be changed

Description

This API is used to set mtu of the netif. This will be called whenever mtu needs to be changed

Parameters:

netif Indicates the lwIP network interface [N/A]

mtu Indicates the new MTU of the network interface. Valid values are 68 to 1500. [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments,

Required Header File

netifapi.h

Note

1. On modifying MTU, MTU change will come into effect immediately. 2. IP packets for existing connection shall also be sent according to new MTU. 3. Ideally, application must ensure that connections are terminated before MTU modification or at init time to avoid side effects, since peer might be expecting different MTU. 4. Effective MSS for existing connection wont change, it might remain same. At runtime it is not suggested to change MTU. 5. Only for new connections, effective MSS shall be used for connection setup

N/A

Related Topics

N/A

Netifapi_stop_queue

Prototype

```
err_t netifapi_stop_queue(struct netif *netif);
```

Purpose

This API is used to set the netif driver status to "Driver Not Ready" state. Driver needs to call this API to intimate the stack that the send buffer in the driver is full.

Description

This API is used to set the netif driver status to "Driver Not Ready" state. Driver needs to call this API to intimate the stack that the send buffer in the driver is full.

Parameters:

netif The lwIP network interface [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments,

Required Header File

netifapi.h

Note

N/A

Related Topics

N/A

Netifapi_wake_queue

Prototype

```
err_t netifapi_wake_queue(struct netif *netif);
```

Purpose

This API is used to set the netif driver status to "Driver Ready" state. This API is called by the driver to inform the stack that the driver send buffer is available to send after a netifapi_stop_queue was called previously

Description

This API is used to set the netif driver status to "Driver Ready" state. This API is called by the driver to inform the stack that the driver send buffer is available to send after a netifapi_stop_queue was called previously

Parameters:

netif The lwIP network interface [N/A]

Return values

ERR_OK: On success

ERR_ARG: On passing invalid arguments,

Required Header File

netifapi.h

Note

N/A

Related Topics

N/A

PTlwIPFuncMemSet_s

Prototype

```
typedef int  (*PTlwIPFuncMemSet_s)(void *pvDest,
                                    unsigned int ulDestMax, int Char, unsigned int ulCount);
```

Description

Secure callback function for Memory Setting.

PTlwIPFuncMemCpy_s

Prototype

```
typedef int  (*PTlwIPFuncMemCpy_s)(void *pvDest,
                              unsigned int ulDestMax, const void *Src, unsigned int ulCount);
```

Description

Secure callback function for Memory Copy.

PTlwIPFuncStrNCpy_s

Prototype

```
 typedef int  (*PTlwIPFuncStrNCpy_s)( char *pcDest, unsigned int ulDestMax,
                                                              const char * pcSrc, unsigned int
ulCount );
```

Description

Secure callback function for Memory N bytes Copy.

PTlwIPFuncStrNCat_s

Prototype

```
 typedef int (*PTlwIPFuncStrNCat_s)( char *pcDest, unsigned int ulDestMax,

                                                               const char *pcSrc, unsigned int
ulCount);
```

Description

Secure callback function for memory N bytes concatenation.

PTlwIPFuncStrCat_s

Prototype

```
 typedef int (*PTlwIPFuncStrCat_s)( char *pcDest, unsigned int ulDestMax,

const char *pcSrc);
```

Description

Secure callback function for memory concatenation.

PTlwIPFuncMemMove_s

Prototype

```
 typedef int (*PTlwIPFuncMemMove_s)(void *pcDest,

                              unsigned int ulDestMax, const void *pcSrc, unsigned int
ulCount);
```

Description

Secure callback function for memory move.

PTlwIPFuncSnprintf_s

Prototype

```
typedef int (*PTlwIPFuncSnprintf_s) (char* pcStrDest, unsigned int ulDestMax,

   unsigned int ulCount, const char* pszFormat, ...);
```

Description

Secure callback function for string to buffer copy.

PTlwIPFuncRand_s

Prototype

```
typedef int (*PTlwIPFuncRand_s)(void);
```

Description

Secure callback function for Random Number Generator.

Lwip_secure_impl_s

Prototype

```
struct lwip_secure_impl_s

{

  PTlwIPFuncMemSet_s     pfMemset_s;

  PTlwIPFuncMemCpy_s    pfMemcpy_s;

  PTlwIPFuncStrNCpy_s    pfStrNCpy_s;

  PTlwIPFuncStrNCat_s     pfStrNCat_s;

  PTlwIPFuncStrCat_s       pfStrCat_s;

  PTlwIPFuncMemMove_s   pfMemMove_s;

  PTlwIPFuncSnprintf_s     pfSnprintf_s;

  PTlwIPFuncRand_s       pfRand;

};
```

:

PTlwIPFuncMemSet_s Secure callback function for Memory Setting

PTlwIPFuncMemCpy_s Secure callback function for Memory Copy

PTlwIPFuncStrNCpy_s Secure callback function for Memory N bytes Copy

PTlwIPFuncStrNCat_s Secure callback function for memory N bytes concatenation

PTlwIPFuncStrCat_s Secure callback function for memory concatenation

PTlwIPFuncMemMove_s Secure callback function for memory move

PTlwIPFuncSnprintf_s Secure callback function for string to buffer copy

PTlwIPFuncRand_s Secure callback function for Random Number Generator

LwIPRegSecSspCbk

Prototype

```
int lwIPRegSecSspCbk(const PTlwIPSecFuncSsp pstSecureFuncSsp);
```

Purpose

Registering the secure call backs.

Description

Setting the register callback.

Parameters:

pstSecureFuncSsp Pointer to structure call back. [N/A]

Return values

- ERR_OK: On success
- ERR_VAL: On failure due to Illegal value

Required Header File

opt.h

Note

N/A

Related Topics

N/A

Securelib

- **Secure SSP Datastructures**
- **Securelib Interface**

Secure SSP Datastructures

- **Lwip_secure_impl_s**

Securelib Interface

- **PTlwIPFuncMemSet_s**
- **PTlwIPFuncMemCpy_s**
- **PTlwIPFuncStrNCpy_s**
- **PTlwIPFuncStrNCat_s**
- **PTlwIPFuncStrCat_s**
- **PTlwIPFuncMemMove_s**
- **PTlwIPFuncSnprintf_s**
- **PTlwIPFuncRand_s**
- **LwIPRegSecSspCbk**

Pbuf_ram_size_set

Prototype

```
u32_t pbuf_ram_size_set(u32_t ram_max_size);
```

Purpose

This API is used to set the maximum buffer size of PBUF_RAM allocation.

Description

This API is used to set the maximum buffer size of PBUF_RAM allocation

Parameters:

ram_max_size maximum ram buffer size allowed. [The ram_max_size must be greater than PBUF_RAM_SIZE_MIN and less than 0x7FFFFFFF.]

Return values

Zero : On failure

Non Zero value: Previous maximum ram buffer size

Required Header File

pbuf.h

Note

None

Related Topics

None

Pbuf_ram_display

Prototype

```
void pbuf_ram_display(void);
```

Purpose

This API is used to display the pbuf RAM details.

Description

This API is used to display the pbuf RAM details.

Parameter

void

Return values

void

Required Header File

pbuf.h

Note

This API is enabled only if the LWIP_DEBUG macro is enabled.

Related Topics

None

Buffer_Interfaces

- **Pbuf_ram_size_set**
- **Pbuf_ram_display**

Sockaddr_in

Prototype

```
struct sockaddr_in

{

 u16_t sin_family;

 u16_t sin_port;

 struct in_addr sin_addr;

 char sin_zero[8];

};
```

:

sin_family Specifies the socket family.

sin_port Specifies the port.

sin_address Specifies the address.

sin_zero Indicates an 6-bit padding.

Socket

- **Socket Structure**
- **Socket Interfaces**

Socket Structure

- **Sockaddr_in**
- **Sockaddr**
- **Sockaddr_ll**
- **Linger**

Socket Interfaces

- **Lwip_accept**
- **Lwip_bind**
- **Lwip_shutdown**
- **Lwip_getpeername**
- **Lwip_getsockname**
- **Lwip_getsockopt**
- **Lwip_setsockopt**
- **Lwip_close**
- **Lwip_connect**
- **Lwip_listen**
- **Lwip_recv**
- **Lwip_read**
- **Lwip_recvfrom**
- **Lwip_send**
- **Lwip_sendto**
- **Lwip_socket**
- **Lwip_write**
- **Lwip_select**
- **Lwip_ioctl**
- **Lwip_fcntl**
- **Lwip_get_conn_info**

Sockaddr

Prototype

```
   struct sockaddr
{
 u16_t sa_family;
 char sa_data[14];
};
```

:

sa_family Specifies the socket family.

sa_data Specifies the address.

Sockaddr_ll

Prototype

```
  struct sockaddr_ll
{
 u16_t sll_family;
 u16_t sll_protocol;
 u32_t sll_ifindex;
 u16_t sll_hatype;
 u8_t  sll_pkttype;
 u8_t  sll_halen;
 u8_t  sll_addr[8];
};
```

:

sll_family Specifies the socket family.

sll_protocol Specifies the Ethernet protocol.

sll_ifindex Specifies the network interface index. Starts from 1.

sll_hatype Specifies the header type.

sll_pkttype Specifies the packet type.

sll_halen Specifies the physical layer address length.

sll_addr Specifies the physical layer address.

Note

You must enable PF_PKT_SUPPORT to use the sockaddr_ll data structure.

Linger

Prototype

```
struct linger
 {
   int l_onoff;
   int l_linger;
 };
```

:

l_onoff Specifies the linger option on/off.

l_linger Specifies the linger time.

Lwip_accept

Prototype

```
int lwip_accept(int s, struct sockaddr *addr, socklen_t *addrlen);
```

Purpose

This API is used to accept a connection on a socket.

Description

This API extracts the first connection request from the queue of pending connections for the listening socket 's', creates a new connected socket, and returns a new file descriptor referring to that socket. The newly created socket is not in the the listening state. The original socket 's' is unaffected by this call.

Parameters:

s Specifies a file descriptor referring to the original input socket. [N/A]

addr Indicates a pointer to the sockaddr structure that identifies the connection. [N/A]

addrlen Indicates the size name structure. [N/A]

Return values

New socket file descriptor: On success

-1: On failure

Required Header File

sockets.h

Note

- This API does not support the PF_PACKET option.

Related Topics

lwip_connect

Lwip_bind

Prototype

```
int lwip_bind(int s, const struct sockaddr *name, socklen_t namelen);
```

Purpose

This API is used to associate a local address or name with a socket.

Description

This API assigns the address specified by name to the socket referred to by the file descriptor 's'. The namelen parameter specifies the size, in bytes, of the address structure pointed to by the name parameter.

Parameters:

s Specifies a file descriptor referring to the original input socket. [N/A]

name Specifies a pointer to the sockaddr structure that identify connection. [N/A]

namelen Specifies the size of the name structure. [N/A]

Return values

0: On success

-1: On failure

Required Header File

sockets.h

Note

N/A

Related Topics

N/A

Lwip_shutdown

Prototype

```
int lwip_shutdown(int s, int how);
```

Purpose

This API is used to shut down socket send and receive operations.

Description

This API is used to shut down socket send and receive operations. lwip_bind() assigns the address specified by name to the socket referred to by the file descriptor 's'.

Parameters:

s Specifies a file descriptor referring to the socket. [N/A]

how Specifies the type of shutdown. [SHUT_RD|SHUT_WR|SHUT_RDWR]

Return values

0: Indicates successful execution.

-1: Indicates a failure.

Required Header File

sockets.h

Note

- Only "SHUT_RDWR" is supported for the "how" parameter in this API. lwIP does not support closing one end of the full-duplex connection.
- This API does not support the PF_PACKET option.
- The listen socket does not support half shutdown.
- lwIP does not support half shutdown. Any half shutdown is treated as full connection shutdown.

Related Topics

N/A

Lwip_getpeername

Prototype

```
int lwip_getpeername (int s, struct sockaddr *name, socklen_t *namelen);
```

Purpose

This API is used to get the name of a connected peer socket.

Description

This API returns the address of the peer connected to the socket 's', in the buffer pointed to by name. Initialize the namelen argument to indicate the amount of space pointed to by the name parameter. On return it contains the actual size, in bytes, of the name returned. The name is truncated if the buffer provided is too small.

Parameters:

s Specifies the file descriptor referring to the connected socket. [N/A]

name Indicates the pointer to the sockaddr structure that identifies the connection. [N/A]

namelen Specifies the size name structure. [N/A]

Return values:

int Indicates successful execution. [0|N/A]

int Indicates a failure. [-1|N/A]

Required Header File

sockets.h

Note

- This API does not support the PF_PACKET option.

Related Topics

lwip_getsockname

Lwip_getsockname

Prototype

```
int lwip_getsockname (int s, struct sockaddr *name, socklen_t *namelen);
```

Purpose

This API is used to get the name of a socket.

Description

This API returns the current address to which the socket 's' is bound, in the buffer pointed to by the name parameter. Initialize the namelen argument to indicate the amount of space(in bytes) pointed to by the name parameter. The returned address is truncated if the buffer provided is too small. In this case, namelen returns a value greater than was supplied to the call.

Parameters:

s Specifies the file descriptor referring to connected socket. [N/A]

name Indicates a pointer to sockaddr structure that identifies the connection. [N/A]

namelen Specifies the size name structure. [N/A]

Return values:

int Indicates successful execution. [0|N/A]

int Indicates a failure. [-1|N/A]

Required Header File

sockets.h

Note

- This API does not support the PF_PACKET option.

Related Topics

lwip_getpeername

Lwip_getsockopt

Prototype

```
int lwip_getsockopt (int s, int level, int optname, void *optval, socklen_t *optlen);
```

Purpose

This API is used to get the options set on a socket.

Description

This API retrieves the value for the option specified by the optname argument for the socket specified by the s parameter. If the size of the optval is greater than optlen, the value stored in the object pointed to by the optval argument is truncated.

Parameters:

s Specifies a socket file descriptor. [N/A]

level Specifies the protocol level at which the option resides. [N/A]

optname Specifies a single option to be retrieved. [N/A]

optval Indicates an address to store option value. [N/A]

optlen Specifies the size of the option value. [N/A]

Return values

0: Indicates successful execution.

-1: Indicates a failure.

Required Header File

sockets.h

Note

1. Supported protocol levels are SOL_SOCKET, IPPROTO_IP, IPPROTO_TCP, and SOL_PACKET.

2. Under SOL_SOCKET the options supported are SO_ACCEPTCONN, SO_BROADCAST, SO_ERROR, SO_KEEPALIVE, SO_SNDTIMEO, SO_RCVTIMEO, SO_RCVBUF, SO_REUSEADDR, SO_REUSEPORT, SO_TYPE, SO_NO_CHECK. For SO_SNDTIMEO, SO_RCVTIMEO, and SO_RCVBUF, the macros LWIP_SO_SNDTIMEO, LWIP_SO_RCVTIMEO, and LWIP_SO_RCVBUF must be defined at compile time. For SO_REUSEADDR and SO_REUSEPORT, the macro SO_REUSE must be defined at compile time.

3. Under IPPROTO_IP the options supported are IP_TTL and IP_TOS.

4. Under IPPROTO_TCP the options supported are TCP_NODELAY, TCP_KEEPALIVE, TCP_KEEPIDLE, TCP_KEEPINTVL, TCP_KEEPCNT, and TCP_QUEUE_SEQ. For TCP_KEEPIDLE, TCP_KEEPINTVL, and TCP_KEEPCNT, the macro LWIP_TCP_KEEPALIVE must be defined at compile time.

5. In BSD linux, optval for SO_SNDTIMEO and SO_RCVTIMEO is struct timeval. In lwIP, the optval for SO_SNDTIMEO and SO_RCVTIMEO is int.

6. Under SOL_PACKET the options supported are SO_RCVTIMEO and SO_RCVBUF,SO_TYPE.

7. IP_HDRINCL is supported only if LWIP_RAW is enabled.

8. IP_MULTICAST_TTL, IP_MULTICAST_IF, and IP_MULTICAST_LOOP are supported only if LWIP_IGMP is enabled.

Related Topics

lwip_setsockopt

Lwip_setsockopt

Prototype

```
int lwip_setsockopt (int s, int level, int optname, const void *optval, socklen_t optlen);
```

Purpose

This API is used to set options on a socket.

Description

This API sets the option specified by the optname parameter, at the protocol level specified by the level parameter, to the value pointed to by the optval for the socket associated with the file descriptor specified by the 's' parameter.

Parameters:

s Specifies a socket file descriptor. [N/A]

level Specifies the protocol level at which the option resides. [N/A]

optname Specifies a single option to set. [N/A]

optval Indicates the address to store the option value. [N/A]

optlen Specifies the size of option value. [N/A]

Return values

0: Indicates successful execution.

-1: Indicates a failure.

Required Header File

sockets.h

Note

1. Supported protocol levels are SOL_SOCKET, IPPROTO_IP, IPPROTO_TCP, and SOL_PACKET.

2. Under SOL_SOCKET the options supported are SO_BROADCAST, SO_KEEPALIVE, SO_SNDTIMEO, SO_RCVTIMEO, SO_RCVBUF, SO_REUSEADDR, SO_REUSEPORT, and SO_NO_CHECK. For SO_SNDTIMEO, SO_RCVTIMEO, SO_RCVBUF, the macros LWIP_SO_SNDTIMEO, LWIP_SO_RCVTIMEO, and LWIP_SO_RCVBUF must be defined at compile time. For SO_REUSEADDR and SO_REUSEPORT, the macro SO_REUSE must be defined at compile time.
3. Under IPPROTO_IP the options supported are IP_TTL and IP_TOS.
4. Under IPPROTO_TCP the options supported are TCP_NODELAY, TCP_KEEPALIVE, TCP_KEEPIDLE, TCP_KEEPINTVL, and TCP_KEEPCNT. For TCP_KEEPIDLE, TCP_KEEPINTVL, and TCP_KEEPCNT, the macro LWIP_TCP_KEEPALIVE must be defined at compile time.
5. In BSD linux, optval for SO_SNDTIMEO and SO_RCVTIMEO is struct timeval. In lwIP, the optval for SO_SNDTIMEO and SO_RCVTIMEO is int.
6. Under SOL_PACKET the option supported are SO_RCVTIMEO and SO_RCVBUF,SO_TYPE.
7. IP_HDRINCL is supported only if LWIP_RAW is enabled.
8. IP_MULTICAST_TTL, IP_MULTICAST_IF, and IP_MULTICAST_LOOP are supported only if LWIP_IGMP is enabled.

Related Topics

lwip_getsockopt

Lwip_close

Prototype

```
int lwip_close(int s);
```

Purpose

This API is used to close the socket.

Description

This API closes the socket file descriptor.

Parameters:

s Specifies a socket file descriptor. [N/A]

Return values

0: Indicates a successful execution.

-1: Indicates a failure.

Required Header File

sockets.h

Note

N/A

Related Topics

N/A

Lwip_connect

Prototype

```
int lwip_connect(int s, const struct sockaddr *name, socklen_t namelen);
```

Purpose

This API is used to initiate a connection on the socket.

Description

This API connects the socket referred to by the file descriptor 's' to the address specified by the name parameter.

Parameters:

s Specifies a socket file descriptor. [N/A]

name Specifies a pointer to the sockaddr structure which identifies the connection. [N/A]

namelen Specifies a size name structure. [N/A]

Return values

0: Indicates a successful execution.

-1: Indicates a failure.

Required Header File

sockets.h

Note

N/A

Related Topics

N/A

Lwip_listen

Prototype

```
int lwip_listen(int s, int backlog);
```

Purpose

This API is used to set a socket to the listen mode.

Description

This API marks the socket which the s parameter refers to as a passive socket. This socket is used to accept incoming connection requests using the lwip_accept() function.

Parameters:

s Specifies a socket file descriptor. [N/A]

backlog Specifies the number of connections in the listen queue. [N/A]

Return values

0: Indicates a successful execution.

-1: Indicates a failure.

Required Header File

sockets.h

Note

- This API does not support the PF_PACKET socket.

Related Topics

N/A

Lwip_recv

Prototype

```
int lwip_recv(int s, void *mem, size_t len, int flags);
```

Purpose

This API is used to receive a message from a connected socket.

Description

This API is used to receive messages from a connection-oriented sockets only, because it does not permit the application to retrieve the source address of the received data.

Parameters:

s Specifies a socket file descriptor. [N/A]

mem Provides a buffer to store the received data. [N/A]

len Specifies the number of bytes of data to receive. [N/A]

flags Specifies the types of message reception. [N/A]

Return values

Number of bytes received: Indicates a successful execution.

-1: Indicates a failure.

Required Header File

sockets.h

Note

1. TCP receive buffer is a list which holds the segment received from the peer. If the application calls the lwip_recv function to get the data, the function gets the first entry from the list and gives it back to application. This function does not recursively get entries from the list to fill the complete user buffer.

2. lwIP updates this receive buffer list when it gets the next expected segment. If there is any out of order segment which is next to the received segment, lwIP merges the segments and puts that as one segment on to the receive buffer list.

Related Topics

lwip_read lwip_recvfrom

Lwip_read

Prototype

```
int lwip_read(int s, void *mem, size_t len);
```

Purpose

This API is used to read bytes from a socket file descriptor.

Description

This API is used on a connection-oriented socket to receive data. If the received message is larger than the supplied memory area, the excess data is silently discarded.

Parameters:

s Specifies a socket file descriptor [N/A]

mem Specifies the buffer to store the received data. [N/A]

len Specifies the number of bytes of data to receive. [N/A]

Return values

Number of bytes received: Indicates a successful execution.

-1: Indicates a failure.

Required Header File

sockets.h

Note

N/A

Related Topics

lwip_recv lwip_recvfrom

Lwip_recvfrom

Prototype

```
int lwip_recvfrom(int s, void *mem, size_t len, int flags, struct sockaddr *from, socklen_t
*fromlen);
```

Purpose

This API is used to receive messages from connected and non-connected sockets.

Description

This API is used to receive messages from a connection-oriented and connectionless sockets because it permits the application to retrieve the source address of received data.

Parameters:

s Specifies the socket file descriptor. [N/A]

mem Indicates a buffer to store the received data. [N/A]

len Specifies the number of bytes of data to receive. [N/A]

flags Specifies the types of message reception. [N/A]

from Indicates a pointer to the sockaddr structure that contains the source address of the received data. [N/A]

fromlen Indicates the size of the from structure. [N/A]

Return values

Number of bytes received: Indicates a successful execution.

-1: Indicates a failure.

Required Header File

sockets.h

Note

N/A

Related Topics

lwip_read lwip_recv

Lwip_send

Prototype

```
int lwip_send(int s, const void *dataptr, size_t size, int flags);
```

Purpose

This API is used to send a message to a connected socket.

Description

This API initiates transmission of a message from the specified socket to its peer. This API will send a message only when the socket is connected.

Parameters:

s Specifies the socket file descriptor. [N/A]

dataptr Specifies a buffer containing message to send. [N/A]

size Specifies the length of the message to send. [N/A]

flags Indicates the types of message transmission. [N/A]

Return values

Number of bytes sent: Indicates a successful execution.

-1: Indicates a failure.

Required Header File

sockets.h

Note

1. UDP and RAW connection can send only a maximum data of length 65000. Sending more than that will get truncated to 65000.

N/A

Related Topics

lwip_write lwip_sendto

Lwip_sendto

Prototype

```
 int lwip_sendto(int s, const void *dataptr, size_t size, int flags, const struct sockaddr
*to, socklen_t tolen);
```

Purpose

This API is used to send a message to connected and non-connected sockets.

Description

This API is used to send messages from a connection-oriented and connectionless sockets. If the socket is in the connectionless mode, the message is sent to the address specified by the to parameter. If the socket is in the connection mode, the destination address in the to parameter is ignored.

Parameters:

s Specifies a socket file descriptor. [N/A]

dataptr Specifies a buffer containing the message to send. [N/A]

size Specifies the length of the message to send. [N/A]

flags Indicates the types of message transmission. [N/A]

to Specifies a pointer to the sockaddr structure that contains the destination address. [N/A]

tolen Specifies the size of the to structure. [N/A]

Return values

Number of bytes sent: Indicates a successful execution.

-1: Indicates a failure.

Required Header File

sockets.h

Note

UDP and RAW connection can send only a maximum data of length 65000. Sending more than that will get truncated to 65000.

Related Topics

lwip_write lwip_send

Lwip_socket

Prototype

```
int lwip_socket(int domain, int type, int protocol);
```

Purpose

This API is used to allocate a socket.

Description

This API is used to create an endpoint for communication and returns a file descriptor.

Parameters:

domain Specifies a protocol family. [N/A]

type Specifies the socket type. [SOCK_RAW|SOCK_DGRAM|SOCK_STREAM]

protocol Specifies the protocol to be used with the socket. [N/A]

Return values

Valid socket file descriptor: Indicates a successful execution.

-1: Indicates a failure.

Required Header File

sockets.h

Note

N/A

Related Topics

N/A

Lwip_write

Prototype

```
int lwip_write(int s, const void *dataptr, size_t size);
```

Purpose

This API is used to write data bytes to a socket file descriptor.

Description

This API is used to send data on connection-oriented sockets.

Parameters:

s Specifies a socket file descriptor. [N/A]

dataptr Specifies a buffer containing the message to send. [N/A]

size Specifies the length of the message to send. [N/A]

Return values

Number of bytes sent: Indicates a successful execution.

-1: Indicates a failure.

Required Header File

sockets.h

Note

N/A

Related Topics

lwip_send lwip_sendto

Lwip_select

Prototype

```
 int lwip_select(int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, struct
timeval *timeout);
```

Purpose

This API is used to monitor multiple file descriptors, waiting until one or more of the file descriptors are ready for I/O operations.

Description

This API is used to examine the file descriptor sets whose addresses are passed in the readset, writeset, and exceptset parameters to see whether some of its descriptors are ready for read or write, or if the descriptors have an exception condition pending.

Parameters:

maxfdp1 Specifies a range of file descriptors. [N/A]

readset Specifies a pointer to struct fd_set, and specifies the descriptor to check for being ready to read. [N/A]

writeset Specifies a pointer to struct fd_set, and specifies the descriptor to check for being ready to write. [N/A]

exceptset Specifies a pointer to struct fd_set, and specifies the descriptor to check for pending error conditions. [N/A]

timeout Specifies a pointer to struct timeval, for timeout application. [N/A]

Return values

Socket file descriptor: Indicates a successful execution.

-1: Indicates a failure.

Required Header File

sockets.h

Note

N/A

Related Topics

N/A

Lwip_ioctl

Prototype

```
int lwip_ioctl(int s, long cmd, void *argp);
```

Purpose

This API is used as a control device.

Description

This API is used as a control device.

Parameters:

s Specifies an open socket file descriptor. [N/A]

cmd Specifies a command to select a control function. [N/A]

argp Specifies additional information, if required. [N/A]

Return values

0: Indicates a successful execution.

-1: Indicates a failure.

Required Header File

sockets.h

Note

- Linux API supports variable argument support. The lwIP API supports only one void * as the third argument.
- This API supports the following options:
- SIOCADDRT: Set IF gateway, soft-route is not support by lwIP yet.
- SIOCGIFADDR: Get ifnet address.
- SIOCGIFFLAGS: Get ifnet flags.
- SIOCSIFFLAGS: Set ifnet flags.
-
  - IFF_UP interface is up.

  - IFF_brcast brcast address valid.
  - IFF_LOOPBACK is a loopback net.
  - IFF_POINTOPOINT is a point-to-point link.
  - IFF_DRV_RUNNING resources allocated.
  - IFF_NOARP no address resolution protocol.
  - IFF_MULTICAST supports multicast.
  - IFF_DYNAMIC dialup device with changing addresses.
  - IFF_DYNAMIC_S dialup device with changing addresses.
  -
-

- This API also supports the following options:
- SIOCGIFADDR: get ifnet address.
- SIOCSIFADDR: set ifnet address.
- SIOCGIFNETMASK: get net addr mask.
- SIOCSIFNETMASK : set net addr mask.
- SIOCSIFHWADDR: set IF mac_address.
- SIOCGIFHWADDR: get IF mac_address
- SIOCGIFNAME: set IF name.
- SIOCSIFNAME: set IF name.
- FIONBIO: set/clear non-blocking i/o.
- For FIONREAD option, argp should point to an application variable of type signed int. FIONREAD is supported if LWIP_SO_RCVBUF is enabled.
- For PF_PACKET sockets SIOCADDRT option is not supported. The option supported are FIONBIO, SIOCGIFADDR, SIOCSIFADDR, SIOCGIFNETMASK, SIOCSIFNETMASK, SIOCSIFHWADDR, SIOCGIFHWADDR, SIOCGIFFLAGS, SIOCSIFFLAGS, SIOCGIFNAME, SIOCSIFNAME, SIOCGIFINDEX

Related Topics

N/A

Lwip_fcntl

Prototype

```
int lwip_fcntl(int s, int cmd, int val);
```

Purpose

This API is used to manipulate file descriptor.

Description

This API is used to manipulate file descriptor.

Parameters:

s socket file descriptor [N/A]

cmd command to select an operation [F_GETFL|F_SETFL|N/A]

val additional flag, to set non-blocking [N/A]

Return values

Postitive value: On success

-1: On failure

Required Header File

sockets.h

Note

1. Function prototype does not support variable arguments like linux fcntl API.
2. Only F_GETFL & F_SETFL commands are supported. For F_SETFL, only O_NONBLOCK is supported for val.
3. PF_PACKET sockets supports the F_SETFL and F_GETFL option.

Related Topics

N/A

Lwip_get_conn_info

Prototype

```
int lwip_get_conn_info (int s, void *conninfo);
```

Purpose

This API is used to get TCP or UDP connection information.

Description

This API is used to get TCP or UDP connection information.

Parameters:

s socket file descriptor [N/A]

conninfo Connection information details of given socket

Return values

0: On success

Negative value: On failure

Required Header File

sockets.h

Note

1. This function called to get TCP or UDP connection information.
2. The void pointer is of type tcpip_conn.

Related Topics

N/A

Lwip_tcp_cfg_param

```
struct lwip_tcp_cfg_param_

{

   u32_t tcp_wnd;

   u32_t tcp_sndlowat;

   u32_t tcp_sndqueuelowat;

   tcpwnd_size_t tcp_snd_buf;

   tcpwnd_size_t initial_cwnd;

   tcpwnd_size_t inital_ssthresh;

   tcpwnd_size_t tcp_snd_queuelen;

   u16_t tcp_mss;

   u16_t tcp_oversize;

#if LWIP_WND_SCALE

   u16_t tcp_wnd_min;

   u8_t tcp_rcv_scale;

#endif

   u8_t tcp_ttl;

   u8_t tcp_maxrtx;

   u8_t tcp_synmaxrtx;

#if LWIP_WND_SCALE

   u16_t pad;

 #else

   u8_t pad;

#endif

};
```

Description

This struct is used to configure the structure members.

:

tcp_wnd Size of a TCP window.

tcp_sndlowat TCP writeable space.

tcp_sndqueuelowat TCP writeable buffs.

tcp_snd_buf TCP sender buffer space.

initial_cwnd TCP initial congestion window. .

inital_ssthresh TCP initial Threshold window.

tcp_mss TCP Maximum segment size.

tcp_oversize Maximum number of bytes that TCP write may allocate.

tcp_wnd_min TCP minimum window value.

tcp_rcv_scale Shift count value for Window scale.

tcp_ttl Time-To-Live value.

tcp_maxrtx Maximum number of retransmissions of data segments.

tcp_synmaxrtx Maximum number of retransmissions of SYN segments.

pad used for structure 32 bit boundary alignment.

Lwip_config_tcp_ttl

Prototype

```
int lwip_config_tcp_ttl(u8_t ttl);
```

Purpose

This API is used to configure TCP ttl value.

Description

This API is used to configure TCP ttl value.

Parameters:

ttl Time to live. [N/A]

Return values

0: On Success

Non Zero value: On Failure

Required Header File

tcp_config.h

Note

This is not an external interface, and does not carry any validation for access. Do not use this API.

Related Topics

Lwip_config_tcp_maxrtx

Prototype

```
int lwip_config_tcp_maxrtx(u8_t maxrtx);
```

Purpose

This API is used to configure TCP maximum number of retransmissions of data segments.

Description

This API is used to configure TCP maximum number of retransmissions of data segments.

Parameters:

maxrtx Maximum number of retransmissions of data segments. [N/A]

Return values

0: On Success

Non Zero value: On Failure

Required Header File

tcp_config.h

Note

This is not an external interface, and does not carry any validation for access. Do not use this API.

Related Topics

Lwip_config_tcp_synmaxrtx

Prototype

```
int lwip_config_tcp_synmaxrtx(u8_t maxrtx);
```

Purpose

This API is used to configure TCP maximum number of retransmissions of SYN segments.

Description

This API is used to configure TCP maximum number of retransmissions of SYN segments.

Parameters:

maxrtx Maximum number of retransmissions of SYN segments. [N/A]

Return values

0: On Success Non Zero value: On Failure

Required Header File

tcp_config.h

Note

This is not an external interface, and does not carry any validation for access. Do not use this API. None

Related Topics

None

Lwip_config_tcp_rcv_scale

Prototype

```
int lwip_config_tcp_rcv_scale(u8_t rcv_scale);
```

Purpose

This API is used to configure the shift count value for Window scale. This API is available only if LWIP_WND_SCALE is enabled.

Description

This API is used to configure the shift count value for Window scale. This API is available only if LWIP_WND_SCALE is enabled.

Parameters:

rcv_scale Shift count value for Window scale. [N/A]

Return values

0: On Success Non Zero value: On Failure

Required Header File

tcp_config.h

Note

This is not an external interface, and does not carry any validation for access. Do not use this API.

Related Topics

Lwip_config_tcp_wnd_min

Prototype

```
int lwip_config_tcp_wnd_min(u16_t wnd);
```

Purpose

This API is used to configure the size of a TCP Minimum window. This API is available only if LWIP_WND_SCALE is enabled.

Description

This API is used to configure the size of a TCP Minimum window. This API is available only if LWIP_WND_SCALE is enabled.

Parameters:

wnd Size of a TCP Minimum window.[N/A]

Return values

0: On Success Non Zero value: On Failure

Required Header File

tcp_config.h

Note

This is not an external interface, and does not carry any validation for access. Do not use this API.

Related Topics

None

Lwip_config_tcp_wnd

Prototype

```
int lwip_config_tcp_wnd(u32_t wnd);
```

Purpose

This API is used to configure the size of a TCP window.

Description

This API is used to configure the size of a TCP window.

Parameters:

wnd Size of a TCP window. [N/A]

Return values

0: On Success Non Zero value: On Failure

Required Header File

tcp_config.h

Note

This is not an external interface, and does not carry any validation for access. Do not use this API.

Related Topics

None

Lwip_config_tcp_mss

Prototype

```
int lwip_config_tcp_mss(u16_t mss);
```

Purpose

This API is used to configure the maximum segment size of TCP.

Description

This API is used to configure the maximum segment size of TCP.

Parameters:

mss Maximum segment size of TCP. [N/A]

Return values

0: On Success Non Zero value: On Failure

Required Header File

tcp_config.h

Note

This is not an external interface, and does not carry any validation for access. Do not use this API.

Related Topics

None

Lwip_config_tcp_snd_buf

Prototype

```
int lwip_config_tcp_snd_buf(tcpwnd_size_t snd_buf);
```

Purpose

This API is used to configure TCP sender buffer space (bytes).

Description

This API is used to configure TCP sender buffer space (bytes).

Parameters:

snd_buf TCP sender buffer space. [N/A]

Return values

0: On Success Non Zero value: On Failure

Required Header File

tcp_config.h

Note

This is not an external interface, and does not carry any validation for access. Do not use this API.

Related Topics

None

Lwip_config_initial_cwd

Prototype

```
int lwip_config_initial_cwd(tcpwnd_size_t cwnd);
```

Purpose

This API is used to configure TCP initial congestion window (bytes).

Description

This API is used to configure TCP initial congestion window (bytes).

Parameters:

cwnd TCP initial congestion window. [N/A]

Return values

0: On Success Non Zero value: On Failure

Required Header File

tcp_config.h

Note

This is not an external interface, and does not carry any validation for access. Do not use this API.

Related Topics

None

Lwip_config_initial_ssthresh

Prototype

```
int lwip_config_initial_ssthresh(tcpwnd_size_t ssthresh);
```

Purpose

This API is used to configure TCP initial threshold window (bytes).

Description

This API is used to configure TCP initial threshold window (bytes).

Parameters:

ssthresh TCP initial Threshold window. [N/A]

Return values

0: On Success Non Zero value: On Failure

Required Header File

tcp_config.h

Note

This is not an external interface, and does not carry any validation for access. Do not use this API.

Related Topics

None

TCP Configuration

- **TCP Configuration API**

TCP Configuration API

- **Lwip_config_tcp_ttl**
- **Lwip_config_tcp_maxrtx**
- **Lwip_config_tcp_synmaxrtx**
- **Lwip_config_tcp_rcv_scale**
- **Lwip_config_tcp_wnd_min**
- **Lwip_config_tcp_wnd**
- **Lwip_config_tcp_mss**
- **Lwip_config_tcp_snd_buf**
- **Lwip_config_initial_cwd**
- **Lwip_config_initial_ssthresh**

Tcpip_init_done_fn

Prototype

```
typedef void (*tcpip_init_done_fn)(void *arg);
```

Purpose

This callback checks whether initialization is done.

Description

This callback checks whether initialization is done.

Parameters:

arg Specifies the argument to pass to tcpip_init_done. [N/A]

Return values

None

Required Header File

tcpip.h

Note

N/A

Related Topics

N/A

TCPIP

- **Datastructures**
- **Tcip Interface**

Detailed Description

This section contains the TCP/IP related interfaces.

Tcpip_init

Prototype

```
void tcpip_init(tcpip_init_done_fn tcpip_init_done, void *arg);
```

Purpose

This API initializes all sub modules and starts the tcpip_thread.

Description

This API initializes all sub modules and starts the tcpip_thread.

Parameters:

tcpip_init_done Specifies the function to call when the tcpip_thread is running and finished initializing. [N/A]

arg Specifies the argument to pass to tcpip_init_done. [N/A]

Return values

None

Required Header File

tcpip.h

Note

N/A

Related Topics

N/A

Tcpip_conn

```
 struct tcpip_conn {

  struct eth_addr dst_mac;

  ip_addr_t src_ip;

  ip_addr_t dst_ip;

  u16_t ipid;

  u16_t srcport;

  u16_t dstport;

  u32_t tcpwin;

  u32_t seqnum;

  u32_t acknum;

  u32_t last_payload_len;

  u32_t tsval;

  u32_t tsecr;

};
```

Description

Stores the TCP/UDP connection information.

:

dst_mac Specifies the destination MAC.

src_ip Specifies the source IP address.

dst_ip Specifies the destination IP address of a TCP/UDP connection. If lwip_connect is not called for a UDP connection, then this field will be set to 0 for that UDP connection.

ipid Retains the same value as identification field in the IP header.

srcport Specifies the source port address.

dstport Specifies the destination port address. If connect is not called for a UDP connection, then this field will be set to 0 for that UDP connection.

tcpwin Specifies the TCP window of the last sent TCP packet. For a UDP connection, this field is set to 0.

seqnum Specifies the TCP sequence number of the last ACKED byte of a TCP connection. For a UDP connection, this field is set to 0.

acknum Specifies the TCP Ack number of the last sent packet in the TCP connection. For a UDP connection, this field is set to 0.

last_payload_len Specifies the UDP/TCP payload length of the last packet sent in the UDP/TCP connection.

tsval Indicates the timestamp value. This field is 0 for both TCP and UDP connections.

tsecr Indicates the timestamp echo reply. This field is 0 for both TCP and UDP connections.

Lwip_set_socket_num

Prototype

```
void lwip_set_socket_num(int socketnum);
```

Purpose

This API is used to configure max socket num in lwip.

Description

This API is used to configure max socket num in lwip.

Parameters:

socketnum Maximum number sockets. [N/A]

Return values

0: On Success

Non Zero value: On Failure

Required Header File

tcpip.h

Note

Call this API before tcpip_init, or it will be failed.

If this API is not invoked, max socket num is set to DEFAULT_LWIP_NUM_SOCKETS, which is defined in lwipopts.h.

Related Topics

N/A

Lwip_get_socket_num

Prototype

```
void lwip_get_socket_num(void);
```

Purpose

This API is used to get max socket num in lwip.

Description

This API is used to get max socket num in lwip.

Parameters:

void

Return values

current max socket number

Required Header File

tcpip.h

Note

N/A .

Related Topics

N/A

Datastructures

- **Tcpip_conn**

Tcip Interface

- **Tcpip_init_done_fn**
- **Tcpip_init**
- **lwip_set_socket_num**
- **lwip_get_socket_num**

Netif_status_callback_fn

Prototype

```
typedef void (*netif_status_callback_fn)(struct netif *netif);
```

Description

Netif status callback.

Netif

Prototype

```
struct netif {

    struct netif *next;


    ip_addr_t ip_addr;

    ip_addr_t netmask;

    ip_addr_t gw;


    netif_input_fn input;

    netif_output_fn output;

    netif_linkoutput_fn linkoutput;


    #if LWIP_NETIF_STATUS_CALLBACK

      netif_status_callback_fn status_callback;

    #endif


    #if LWIP_NETIF_LINK_CALLBACK

      netif_status_callback_fn link_callback;

    #endif


    #if LWIP_NETIF_REMOVE_CALLBACK

      netif_status_callback_fn remove_callback;

    #endif


    void *state;


    drv_send_fn drv_send;

    drv_set_mac_fn drv_set_mac;
```

```
drv_get_mac_fn drv_get_mac;


#if LWIP_DHCP

  struct dhcp *dhcp;

   #if LWIP_DHCPS

     struct dhcps *dhcps;

   #endif

#endif


#if LWIP_AUTOIP

  struct autoip *autoip;

#endif


#if LWIP_NETIF_HOSTNAME

  char*  hostname;

#endif


u16_t mtu;

u8_t hwaddr_len;

u8_t hwaddr[NETIF_MAX_HWADDR_LEN];

u16_t link_layer_type;

u8_t flags;

char name[2];

u8_t num;


#if LWIP_SNMP

 u8_t link_type;

 u32_t link_speed;

 u32_t ts;

 u32_t ifinoctets;

 u32_t ifinucastpkts;

 u32_t ifinnucastpkts;
```

```
    u32_t ifindiscards;

    u32_t ifoutoctets;

    u32_t ifoutucastpkts;

    u32_t ifoutnucastpkts;

    u32_t ifoutdiscards;

   #endif


   #if LWIP_IGMP

    netif_igmp_mac_filter_fn igmp_mac_filter;

   #endif


   #if LWIP_NETIF_HWADDRHINT

    u8_t *addr_hint;

   #endif


   #if ENABLE_LOOPBACK

    struct pbuf *loop_first;

    struct pbuf *loop_last;

     #if LWIP_LOOPBACK_MAX_PBUFS

      u16_t loop_cnt_current;

     #endif

   #endif


};
```

:

**next* Pointer to next in linked list

ip_addr IP_add configuration in network byte order *netmask* Netmask for the IP

gw Gateway

input This function is called by the network device driver to pass a packet up the TCP/IP stack.

output This function is called by the IP module when it wants to send a packet on the interface. This function typically first resolves the hardware address, then sends the packet.

linkoutput This function is called by the ARP module when it wants to send a packet on the interface. This function outputs the pbuf as-is on the link medium.

status_callback This function is called when the netif state is set to up or down

link_callback This function is called when the netif link is set to up or down

remove_callback This function is called when the netif has been removed

**state* This field can be set by the device driver and could point to state information for the device.

drv_send This function is called when lwIP want to send a packet to interface.

drv_set_hwaddr This function is called when lwIP want to set the mac_address of the interface.

**dhcp* The DHCP client state information for this netif

**dhcps* DHCP Server Informarion for this netif

**autoip* The AutoIP client state information for this netif

hostname The hostname for this netif, NULL is a valid value

mtu Maximum transfer unit (in bytes)

hwaddr_len Number of bytes used in hwaddr

hwaddr[NETIF_MAX_HWADDR_LEN] Link level hardware address of this interface

link_layer_type Link layer type, ethernet or wifi

flags flags (see NETIF_FLAG_ above)

name[2] Descriptive abbreviation

num Number of this interface

link_type Link type (from "snmp_ifType" enum from snmp.h)

link_speed (estimate) Link speed

ts Timestamp at last change made (up/down)

ifinoctets counters

ifinucastpkts counters

ifinnucastpkts counters

ifindiscards counters

ifoutoctets counters

ifoutucastpkts counters

ifoutnucastpkts counters

ifoutdiscards counters

igmp_mac_filter This function could be called to add or delete a entry in the multicast filter table of the ethernet MAC

**addr_hint* Hardware type hint

**loop_first* List of packets to be queued for ourselves.

**loop_last* List of packets to be queued for ourselves.

loop_cnt_current pbuf count

Netif_set_status_callback

Prototype

```
 void netif_set_status_callback(struct netif *netif, netif_status_callback_fn
status_callback);
```

Purpose

Set callback to be called when interface is brought up/down.

Description

Set callback to be called when interface is brought up/down.

Parameters:

netif Netif Structure. [N/A]

netif_status_callback_fn Indicates the Netif set status call function. [N/A]

Required Header File

netif.h

Note

None

Related Topics

None

Netif_set_remove_callback

Prototype

```
 void netif_set_remove_callback(struct netif *netif, netif_status_callback_fn
remove_callback);
```

Purpose

Set callback to be called when the interface has been removed.

Description

Set callback to be called when the interface has been removed.

Parameters:

netif Netif Structure. [N/A]

netif_status_callback_fn Indicates the Netif set status call function. [N/A]

Required Header File

netif.h

Note

None

Related Topics

None

Index

INDEX