MiniTCP v0.12.4.27

Programmer's reference manual

Table of contents

Functions

General overview

MiniTCP is a lightweight, open-source TCP/IP stack designed for 16-bit DOS systems. It is implemented as a C library. It is designed to work with a standard DOS ethernet packet driver. It can also be modified quite easily to work with some other method of ethernet support besides packet drivers. If you plan to just use a regular DOS packet driver, you can skip the rest of this page.

To get rid of the packet driver specific code, remove the ”**static void far interrupt callback**” function. You must also remove the code in “**void pkt_send**” and replace it with code suitable to the method with which you will access ethernet.

When you receive a new ethernet packet, you must copy the raw packet data into the structure at **packet[pktidx]**. If the total size of this packet if greater than 1514 bytes, it is too large and you must drop it.

After filling the structure with the new packet, call the function “**static void far have_new_packet**”. This lets the TCP/IP stack know that there has been a new packet placed in it's buffer.

int16 tcpInit(uint8 packet_int, uint8 tcp_count, uint16 tcp_buffer_size);

Description:

Initializes the MiniTCP library. This must be called before using any sockets. You must set local network settings ***BEFORE*** *calling this function by using either* ***load_config()*** *or manually using* ***tcp_changeip()****,* ***tcp_changesubnet()****,* ***tcp_changegateway()****, and* ***tcp_changedns()****.*

Parameters:

- packet_int – Interrupt number of the installed packet driver for MiniTCP to use. It is almost always 0x60.
- tcp_count - The number of TCP socket structures to be allocated.
- tcp_buffer_size – Bytes of memory each TCP socket should allocate for incoming data.

Return value on success:

- MINITCP_STATUS_OK

Return value on error:

- MINITCP_ERR_PKT_DRV – Could not get handle from packet driver.
- MINITCP_ERR_OUT_OF_MEM – Unable to allocate enough memory for sockets.
- MINITCP_ERR_TOO_MANY – tcp_count value given is greater than the value of TCPMAXSOCKS when the library was compiled.

See also:

tcpShutdown, **load_config**, **tcp_changeip**, **tcp_changesubnet**, **tcp_changegateway**, **tcp_changedns**

void tcpShutdown();

Description:

*Disables the MiniTCP library. Should be used before exiting a program if MiniTCP was previously initialized via **tcpInit**.*

Parameters:

- None

See also:

tcpInit

int16 load_config();

Description:

*This can be used to configure MiniTCP settings by reading them from the file **MINITCP.INI**. Before using **tcpInit**, the program must configure the network settings either with this function, or manually setting value with **tcp_changeip**, **tcp_changesubnet**, **tcp_changegateway**, and **tcp_changedns**.*

Parameters:

- None

Return value on success:

- MINITCP_STATUS_OK

Return value on error:

- MINITCP_ERR_FILE – The config file could not be opened.

See also:

tcpInit, **tcp_changeip**, **tcp_changesubnet**, **tcp_changegateway**, **tcp_changedns**

void process_packet();

Description:

This function must be called regularly to make MiniTCP process buffered incoming ethernet packets. Once all incoming packet buffers are used, MiniTCP will simply drop new incoming packets. Call it often to avoid this situation. It is safe to attach this to the system timer interrupt. It will never be re-entrant, because it disables interrupts until it is finished.

Parameters:

- None

void tcp_changeip(uint8 *string);
void tcp_changesubnet(uint8 *string);
void tcp_changegateway(uint8 *string);
void tcp_changedns(uint8 *string);

Description:

*These can be used to configure MiniTCP local network settings manually. They should be used before using **tcpInit**.*

Parameters:

- *string – Pointer to a null-terminated string containing an appropriate dotted-quad. (i.e. “192.168.1.10”)

See also:

tcpInit

uint16 dns_query(uint8 *hostname, uint8 *ipdest, uint16 *id_token);

Description:

*Initiates a non-blocking hostname resolution query. After initiating this query, the program should use the value returned into the variable pointed at by **id_token** with the **dns_query_by_token** function repeatedly to check for an answer from the DNS server.*

Parameters:

- *hostname – Pointer to a null-terminated string containing the hostname which the program needs to resolve. It is safe to give a hostname string given of a valid IP address in dotted-quad format. It will simply be instantly "resolved" internally without sending a packet to the DNS server.
- *ipdest – Pointer to a 4-byte buffer supplied by your program in which this call will immediately place the resolved IP address if it was already in our local cache. If this happens, the function's return value is non-zero.

Return value on success:

- Non-zero – The hostname supplied was already in the local cache or was a valid dotted-quad already. The resolved IP address was placed in a 4-byte buffer which you supplied a pointer to as ***ipdest**. There is no need for any further calls to complete the operation.
- Zero – The DNS query has been sent. Use the value placed in ***id_token** in calls to the **dns_query_by_token** until it returns an IP address, or you decide to give up.

See also:

dns_query_by_token, dns_resolve

uint16 dns_query_by_token(uint16 *id_token, uint8 *ipdest);

Description:

Checks if a previous query identified by the token has received a response.

Parameters:

- *id_token – Pointer to a uint16 containing the token to use. Should be the same pointer you gave in the **dns_query** function call.
- *ipdest – Pointer to a 4-byte buffer supplied by your program in which this call will place the resolved IP address, if available. If this happens, the function's return value is non-zero.

Return value on success:

- Non-zero – The hostname was successfully resolved to an IP. The resolved IP address was placed in a 4-byte buffer which you supplied a pointer to as ***ipdest**.
- Zero – There has not yet been a response to the query you specified.

See also:

dns_query, dns_resolve

uint16 dns_resolve(uint8 *hostname, uint8 *ipdest, uint16 timeout);

Description:

Performs a blocking call to resolve a hostname or IP address dotted-quad, and if successful, places the result into the 4-byte buffer you specified a pointer to as ****ipdest****. If MiniTCP was unable to resolve the given hostname within the time specified (in milliseconds) in* ***timeout****, it will return a value of zero and will not modify the data at* ****ipdest****.*

Parameters:

- *hostname – Pointer to a null-terminated string containing the hostname which the program needs to resolve. It is safe to give a hostname string given of a valid IP address in dotted-quad format. It will simply be instantly “resolved” internally without sending a packet to the DNS server.
- *ipdest – Pointer to a 4-byte buffer supplied by your program in which this call will place the resolved IP address, if resolution was successful. If this happens, the function's return value is non-zero.
- timeout – Specifies the maximum amount of time to wait for a DNS result before failing, in milliseconds.

Return value on success:

- Non-zero – The hostname was successfully resolved to an IP. The resolved IP address was placed in a 4-byte buffer which you supplied a pointer to as ***ipdest**.
- Zero – The hostname resolution attempt failed. ***ipdest** has not been modified.

See also:

dns_query, dns_query_by_token

uint8 *tcp_localip();
uint8 *tcp_localsubnet();
uint8 *tcp_localgateway();
uint8 *tcp_localdns();
uint8 *tcp_remoteip(uint16 handle);

Description:

These functions return a pointer to the appropriate 4-byte value.

Parameters:

- None

Return value:

- A uint8 pointer to a 4-byte value of the appropriate type.

uint16 tcp_remoteport(uint16 handle);
uint16 tcp_localport(uint16 handle);

Description:

These functions return a uint16 according to their names, the remote or local port number for a specific TCP socket.

Parameters:

- handle – The handle value of the TCP socket that was given when it was created.

Return value:

- A uint16 with the requested data.

int16 tcp_listen(uint16 local_port);

Description:

Opens a new TCP socket as a listener for incoming connections on the given port. Returns the handle value of the new socket if successful, otherwise returns -1.

Parameters:

- local_port – The local port number this socket should listen for connections on.

Return value:

- -1 on failure.
- Any other value is the handle value of the new socket.

See also:

tcp_connect, tcp_connect_2, tcp_close

int16 tcp_connect(uint8 *remote_ip, uint16 remote_port);

Description:

Opens a new outgoing TCP socket on the given port. ****remote_ip*** *is a pointer to a null-terminated string with the target IP address in dotted-quad format. (i.e. "192.168.1.10") Returns the handle value of the new socket if successful, otherwise returns -1.*

Parameters:

- *remote_ip – Pointer to null-terminated string with IP in dotted-quad format.
- remote_port – The local port number this socket should listen for connections on.

Return value:

- -1 on failure.
- Any other value is the handle value of the new socket.

See also:

tcp_connect_2, tcp_listen, tcp_close

int16 tcp_connect_2(uint8 *ipaddr, uint16 remote_port);

Description:

Opens a new outgoing TCP socket on the given port. ****ipaddr*** *is a pointer to a 4-byte value in memory of the target IP address. Returns the handle value of the new socket if successful, otherwise returns -1.*

Parameters:

- *ipaddr – Pointer to 4-byte IP address in memory.
- remote_port – The local port number this socket should listen for connections on.

Return value:

- -1 on failure.
- Any other value is the handle value of the new socket.

See also:

tcp_connect_2, tcp_listen, tcp_close

uint16 tcp_read(uint16 handle, uint8 *dest, uint16 maxlen);

Description:

*Reads data from incoming data buffer of an open TCP socket specified by **handle**. Specify the maximum length of data that the buffer given in ***dest** can hold. After data is transferred to ***dest**, it is removed from the socket buffer.*

Parameters:

- handle – Handle number of open TCP socket.
- *dest – Pointer to buffer where the data should be copied to.
- maxlen – The maximum size in bytes that the given buffer can hold.

Return value:

- Amount of data actually copied to destination buffer. Can range from 0 to value given in **maxlen**.

See also:

tcp_peek, **tcp_send_data**

uint16 tcp_peek(uint16 handle, uint8 *dest, uint16 maxlen);

Description:

Reads data from incoming data buffer of an open TCP socket specified by ***handle****. Specify the maximum length of data that the buffer given in* ****dest*** *can hold. This call is identical to* ***tcp_read****, except that after the data is copied to the destination buffer, it remains in the TCP socket's buffer.*

Parameters:

- handle – Handle number of open TCP socket.
- *dest – Pointer to buffer where the data should be copied to.
- maxlen – The maximum size in bytes that the given buffer can hold.

Return value:

- Amount of data actually copied to destination buffer. Can range from 0 to value given in **maxlen**.

See also:

tcp_read, **tcp_send_data**

uint16 tcp_send_data(uint16 handle, uint8 *src, uint16 len);

Description:

*Sends data to remote peer on an open TCP socket specified by **handle**. Specify the length of data to send from the ***src** pointer. Your program is responsible for limiting the size to send, which should not exceed 1400 bytes. Returns amount of data sent. If MiniTCP was not able to immediately send the data, it returns zero and you must try again.*

Parameters:

- handle – Handle number of open TCP socket.
- *src – Pointer to buffer where the data to send is.
- len – The size in bytes to send from that pointer.

Return value:

- Amount of data actually sent. Can range from 0 to value given in **len**.

int16 tcp_established(uint16 handle);

Description:

Use this to determine if there is a currently active connection on the specified TCP socket handle.

Parameters:

- handle – Handle number of open TCP socket.

Return value:

- Non-zero if socket is connected.
- Zero if it is disconnected.

void tcp_close(uint16 handle);

Description:

*Closes a TCP socket connection specified by **handle**. It is safe to call **tcp_close** on a socket that was already disconnected.*

Parameters:

- handle – Handle number of open TCP socket.

void udp_send_data(uint8 *remote_ip, uint16 remote_port, uint16 local_port, uint8 *data, uint16 len);

Description:

Sends data to remote peer in via UDP. ****remote_ip*** *must be a pointer to a 4-byte IP address in memory,* ***not*** *a dotted-quad string.* ***ip_convert*** *can be used to convert a dotted-quad format string to this 4-byte form.*

Parameters:

- *remote_ip – Pointer to 4-byte remote IP to send to.
- remote_port – Port number to send to remote peer on.
- local_port – Local port number.
- *data – Pointer to buffer of data to send.
- Len – Length of data in ***data** outgoing buffer in bytes. Limit this to 1400 or less.

See also:

udp_hook_recv

void udp_hook_recv(void far *callback_ptr);

Description:

Sets a callback pointer in your program that will be used whenever there is a new incoming UDP packet. See the example program ***UDPTEST.C*** *for details on how this callback function works. UDP receive can be disabled again later by passing* ***(void far *)NULL****.*

NOTE: *MiniTCP will never forward incoming UDP packets that are addressed to port 53. This port is reserved for the internal DNS resolution feature.*

Parameters:

- *callback_ptr – A far pointer to your program's UDP receive callback function.

void ipconvert(uint8 *string, uint8 *dest);

Description:

Converts a null-terminated string in ****string*** *containing an IP address in dotted-quad format (i.e. "192.168.1.10") to a 4-byte value and stores it in* ****dest****.*

Parameters:

- *string – Pointer to null-terminated dotted-quad format IP address string.
- *dest – Pointer to store the resulting 4-byte value

void printip(uint8 *src);

Description:

Converts a 4-byte IP address value to dotted-quad format and prints it to the screen.

Parameters:

- *src – Pointer to 4-byte IP address value.

See also:

sprintip

uint16 sprintip(uint8 *dest, uint8 *src);

Description:

Converts a 4-byte IP address value to dotted-quad format and stores the resulting null-terminated string to the pointer supplied by ****dest****.*

Parameters:

- *dest – Pointer to which the output string will be stored.
- *src – Pointer to 4-byte IP address value.

Return value:

Returns the length of the resulting string in byte.

See also:

printip

void attach_timer(uint32 *timerptr);

Description:

*This function causes the 32-bit value in ***timerptr** to start being updated by MiniTCP in real-time. It's value measured in millisends.*

Parameters:

- *timerptr – Pointer to your 32-bit timer variable.

See also:

unattach_timer

void unattach_timer();

Description:

This function causes the 32-bit value in ****timerptr*** *to stop being updated.*

Parameters:

- None.

See also:

attach_timer