sockethelpers

Generated by Doxygen 1.8.1.2

Sat Aug 31 2013 21:40:16

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

Data Structure Documentation

3.1 ServerTag Struct Reference

Struct for passing to server threads.

`#include <socket_helpers_server.h>`

Data Fields

- int c_socket

3.1.1 Detailed Description

Contains a file descriptor for the connected socket, as the server obviously needs to know this.

3.1.2 Field Documentation

3.1.2.1 int ServerTag::c_socket

File descriptor for the connected socket

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

- socket_helpers_server.h

Chapter 4

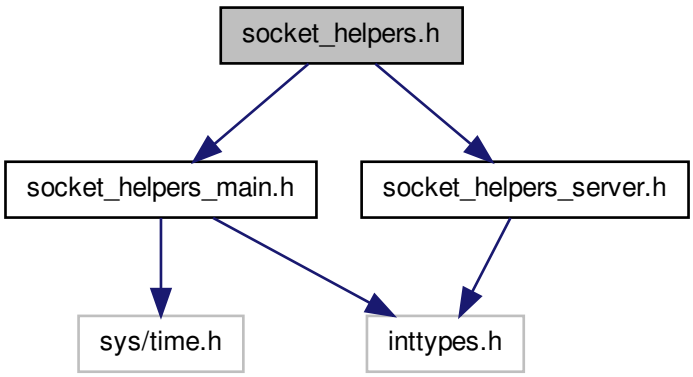
File Documentation

4.1 socket_helpers.h File Reference

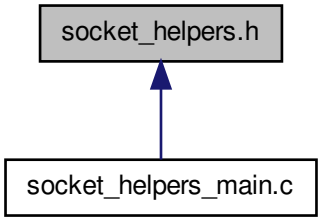
Interface to socket helper library.

```
#include "socket_helpers_main.h"
#include "socket_helpers_server.h"
```

Include dependency graph for socket_helpers.h:



This graph shows which files directly or indirectly include this file:



4.1.1 Detailed Description

Interface to socket helper library.

Author

Paul Griffiths

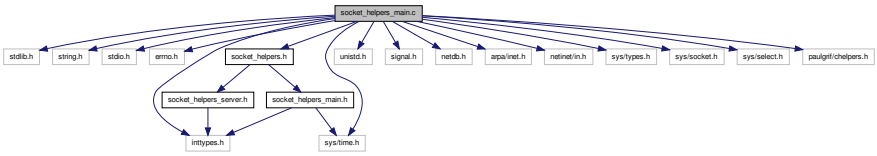
Copyright

4.2 socket_helpers_main.c File Reference

Implementation of main socket helper functions.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <inttypes.h>
#include <unistd.h>
#include <signal.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <sys/select.h>
#include <paulgrif/chelpers.h>
#include "socket_helpers.h"
```

Include dependency graph for socket_helpers_main.c:



Macros

- #define MAX_BUFFER_SIZE 1024

  Maximum character buffer size.

Functions

- ssize_t socket_readline (const int socket, char ∗buffer, const size_t max_len)

  Reads an \r\n terminated line from a socket.
- ssize_t socket_readline_timeout (const int socket, char ∗buffer, const size_t max_len, struct timeval ∗time_out)

  Reads an \r\n terminated line from a socket with timeout.
- ssize_t socket_writeline (const int socket, const char ∗buffer, const size_t max_len)

  Writes a line to a socket.
- uint16_t port_from_string (const char ∗port_str)

  Extracts a valid TCP/UDP port from a string.
- int conn_socket_from_string (const char ∗host, const char ∗port)

  Creates a connected sock from a hostname and port.
- void ignore_sigpipe (void)

  Ignores the SIGPIPE signal.

4.2.1 Detailed Description

Author

Paul Griffiths

Copyright

Copyright 2013 Paul Griffiths. Distributed under the terms of the GNU General Public License. http://www.gnu.org/licenses/

4.2.2 Function Documentation

4.2.2.1 int conn_socket_from_string (const char ∗ *host*, const char ∗ *port*)

Parameters

host	A string containing the hostname to which to connect.
port	A string containing the port to which to connect.

Returns

The file descriptor of the connected socket on success, or -1 on failure.

4.2.2.2 void ignore_sigpipe (void)

The `write()` system call will, when writing to a closed socket, elicit an RST (reset) flag. A second `write()` system call will trigger a `SIGPIPE` signal to be raised. The default action of `SIGPIPE` is to terminate the program, with no error message, which is not desirable. If we want to do anything special when `SIGPIPE` is triggered, we could set up a handler, but if we don't, then ignoring `SIGPIPE` is fine, provided our socket functions respond appropriately to the condition (`write()` will return `EPIPE` after an ignored `SIGPIPE` signal).

4.2.2.3 uint16_t port_from_string (const char * *port_str*)

Parameters

port_str	The string from which to extract

Returns

The port number on success, or zero if `port_str` does not contain a valid TCP/UDP port (port 0 is reserved and cannot be used).

4.2.2.4 ssize_t socket_readline (const int *socket*, char * *buffer*, const size_t *max_len*)

The function will not overwrite the buffer, so `max_len` should be the size of the whole buffer, and function will at most write `max_len - 1` characters plus the terminating `\0`. Any terminating CR or LF characters will be stripped.

Parameters

socket	File description of the socket
buffer	The buffer into which to read
max_len	The maximum number of characters to read, including the terminating `\0`.

Returns

The number of characters read, or -1 on encountering an error.

4.2.2.5 ssize_t socket_readline_timeout (const int *socket*, char * *buffer*, const size_t *max_len*, struct timeval * *time_out*)

Behaves the same as socket_readline(), except it will time out if no input is available on the socket after the specified time. Any terminating CR or LF characters will be stripped.

Parameters

socket	File description of the socket
buffer	The buffer into which to read
max_len	The maximum number of characters to read, including the terminating `\0`.
time_out	A pointer to a `timeval` struct containing the timeout period. Note that some implementations of `select()` may alter this variable, so the calling function should consider it unusable after return. In addition, on such an implementation, the value will specify the cumulative timeout period over the entire read line operation, rather than resetting after reading each character.

Returns

The number of characters read, or -1 on encountering an error.

4.2.2.6 ssize_t socket_writeline (const int *socket*, const char * *buffer*, const size_t *max_len*)

The function adds a network-standard terminating CRLF, so the provided string should not normally end in any newline characters.

Parameters

socket	File description of the socket
buffer	The buffer from which to write.
max_len	The maximum number of characters to write to the buffer. Due to the addition of CRLF, `max_len + 2` characters may actually be written.

Returns

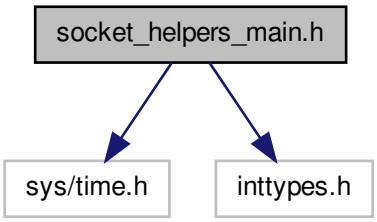
The number of characters written, or -1 on encountering an error.

4.3 socket_helpers_main.h File Reference

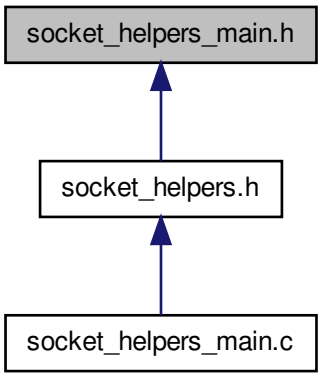
Interface to main socket helper functions.

```
#include <sys/time.h>
#include <inttypes.h>
```

Include dependency graph for socket_helpers_main.h:



This graph shows which files directly or indirectly include this file:



Functions

- ssize_t socket_readline (const int l_socket, char *buffer, const size_t max_len)

  Reads an \r\n terminated line from a socket.
- ssize_t socket_readline_timeout (const int l_socket, char *buffer, const size_t max_len, struct timeval *time_out)

  Reads an \r\n terminated line from a socket with timeout.
- ssize_t socket_writeline (const int l_socket, const char *buffer, const size_t max_len)

  Writes a line to a socket.
- uint16_t port_from_string (const char *port_str)

  Extracts a valid TCP/UDP port from a string.
- int conn_socket_from_string (const char *host, const char *port)

  Creates a connected sock from a hostname and port.
- void ignore_sigpipe (void)

  Ignores the SIGPIPE signal.

4.3.1 Detailed Description

Author

Paul Griffiths

Copyright

4.3.2 Function Documentation

4.3.2.1 int conn_socket_from_string (const char * *host*, const char * *port*)

Parameters

host	A string containing the hostname to which to connect.
port	A string containing the port to which to connect.

Returns

The file descriptor of the connected socket on success, or -1 on failure.

4.3.2.2 void ignore_sigpipe (void)

The `write()` system call will, when writing to a closed socket, elicit an RST (reset) flag. A second `write()` system call will trigger a `SIGPIPE` signal to be raised. The default action of `SIGPIPE` is to terminate the program, with no error message, which is not desirable. If we want to do anything special when `SIGPIPE` is triggered, we could set up a handler, but if we don't, then ignoring `SIGPIPE` is fine, provided our socket functions respond appropriately to the condition (`write()` will return `EPIPE` after an ignored `SIGPIPE` signal).

4.3.2.3 uint16_t port_from_string (const char * *port_str*)

Parameters

port_str	The string from which to extract

Returns

The port number on success, or zero if `port_str` does not contain a valid TCP/UDP port (port 0 is reserved and cannot be used).

4.3.2.4 ssize_t socket_readline (const int *socket,* char * *buffer,* const size_t *max_len*)

The function will not overwrite the buffer, so `max_len` should be the size of the whole buffer, and function will at most write `max_len - 1` characters plus the terminating `\0`. Any terminating CR or LF characters will be stripped.

Parameters

socket	File description of the socket
buffer	The buffer into which to read
max_len	The maximum number of characters to read, including the terminating `\0`.

Returns

The number of characters read, or -1 on encountering an error.

4.3.2.5 ssize_t socket_readline_timeout (const int *socket,* char * *buffer,* const size_t *max_len,* struct timeval * *time_out*)

Behaves the same as socket_readline(), except it will time out if no input is available on the socket after the specified time. Any terminating CR or LF characters will be stripped.

Parameters

socket	File description of the socket
buffer	The buffer into which to read
max_len	The maximum number of characters to read, including the terminating `\0`.

time_out	A pointer to a `timeval` struct containing the timeout period. Note that some implementations of `select()` may alter this variable, so the calling function should consider it unusable after return. In addition, on such an implementation, the value will specify the cumulative timeout period over the entire read line operation, rather than resetting after reading each character.

Returns

The number of characters read, or -1 on encountering an error.

4.3.2.6 ssize_t socket_writeline (const int *socket*, const char * *buffer*, const size_t *max_len*)

The function adds a network-standard terminating CRLF, so the provided string should not normally end in any newline characters.

Parameters

socket	File description of the socket
buffer	The buffer from which to write.
max_len	The maximum number of characters to write to the buffer. Due to the addition of CRLF, `max_len + 2` characters may actually be written.

Returns

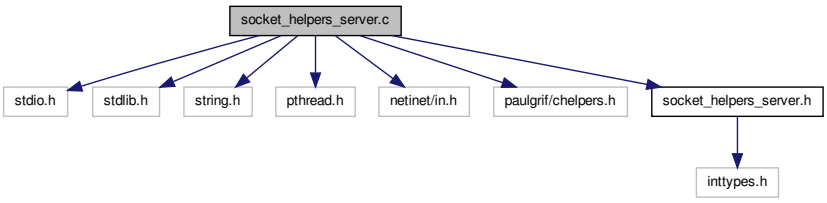
The number of characters written, or -1 on encountering an error.

4.4 socket_helpers_server.c File Reference

Implementation of server helper functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <netinet/in.h>
#include <paulgrif/chelpers.h>
#include "socket_helpers_server.h"
```

Include dependency graph for socket_helpers_server.c:



Macros

- #define IPV6

  Create an IPv6 rather than IPv4 listening socket.

Functions

- int create_tcp_server_socket (const uint16_t listening_port)

  Creates a TCP listening socket.
- int start_threaded_tcp_server (const int listening_socket, void *(*sfunc)(void *))

  Starts an active server.

4.4.1 Detailed Description

Implementation of server helper functions.

Author

Paul Griffiths

Copyright

4.4.2 Function Documentation

4.4.2.1 int create_tcp_server_socket (const uint16_t *listening_port*)

The function creates an IPv4 socket by default, but creates an IPv6 socket if the IPV6 preprocessor macro is defined.

Parameters

listening_port	The port the socket should listen on

Returns

The file descriptor of the created listening socket on success, or -1 on encountering an error.

4.4.2.2 int start_threaded_tcp_server (const int *listening_socket,* void *(*)(void *) *sfunc*)

Connections are passed to a new server thread.

Parameters

listening_socket	A file descriptor for a listening socket.
sfunc	A pointer to a server thread function. The function should return a pointer to void and accept a single pointer to void as an argument, which should be interpreted as a pointer to a ServerTag struct.

Returns

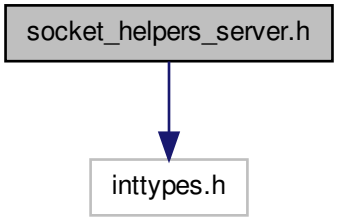
Returns non-zero on encountering an error. The server runs in an infinite loop, and this function will not return unless an error is countered.

4.5 socket_helpers_server.h File Reference

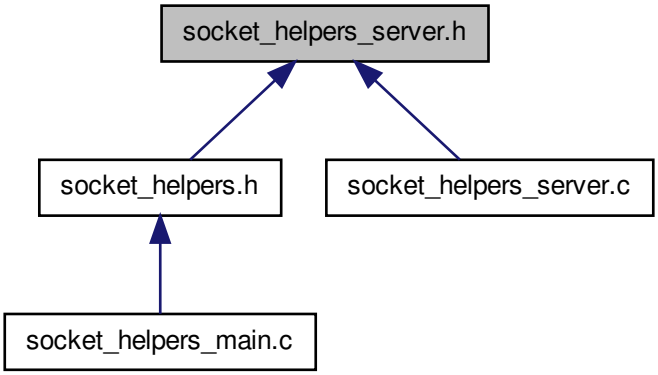
Interface to server helper functions.

```
#include <inttypes.h>
```

Include dependency graph for socket_helpers_server.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct ServerTag

  Struct for passing to server threads.

Typedefs

- typedef struct ServerTag ServerTag

  Struct for passing to server threads.

Functions

- int create_tcp_server_socket (const uint16_t listening_port)

  Creates a TCP listening socket.

- int start_threaded_tcp_server (const int listening_socket, void ∗(∗sfunc)(void ∗))

  Starts an active server.

4.5.1 Detailed Description

Interface to server helper functions.

Author

Paul Griffiths

Copyright

4.5.2 Typedef Documentation

4.5.2.1 typedef struct ServerTag ServerTag

Contains a file descriptor for the connected socket, as the server obviously needs to know this.

4.5.3 Function Documentation

4.5.3.1 int create_tcp_server_socket (const uint16_t *listening_port*)

The function creates an IPv4 socket by default, but creates an IPv6 socket if the IPV6 preprocessor macro is defined.

Parameters

listening_port	The port the socket should listen on

Returns

The file descriptor of the created listening socket on success, or -1 on encountering an error.

4.5.3.2 int start_threaded_tcp_server (const int *listening_socket,* void ∗(∗)(void ∗) *sfunc*)

Connections are passed to a new server thread.

Parameters

listening_socket	A file descriptor for a listening socket.
sfunc	A pointer to a server thread function. The function should return a pointer to void and accept a single pointer to void as an argument, which should be interpreted as a pointer to a ServerTag struct.

Returns

Returns non-zero on encountering an error. The server runs in an infinite loop, and this function will not return unless an error is countered.

Index