

echoserver

Generated by Doxygen 1.8.1.2

Fri Aug 30 2013 19:58:09



# Contents

<b>1</b>	<b>Data Structure Index</b>	<b>1</b>
1.1	Data Structures . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Data Structure Documentation</b>	<b>5</b>
3.1	ServerTag Struct Reference . . . . .	5
3.1.1	Detailed Description . . . . .	5
3.1.2	Field Documentation . . . . .	5
3.1.2.1	c_socket . . . . .	5
3.2	ThreadCount Struct Reference . . . . .	5
3.2.1	Detailed Description . . . . .	6
3.2.2	Field Documentation . . . . .	6
3.2.2.1	count . . . . .	6
3.2.2.2	mutex . . . . .	6
<b>4</b>	<b>File Documentation</b>	<b>7</b>
4.1	echo_server.c File Reference . . . . .	7
4.1.1	Detailed Description . . . . .	8
4.1.2	Macro Definition Documentation . . . . .	8
4.1.2.1	MAX_BUFFER_LEN . . . . .	8
4.1.3	Function Documentation . . . . .	8
4.1.3.1	echo_server . . . . .	8
4.1.4	Variable Documentation . . . . .	8
4.1.4.1	time_out_msg . . . . .	8
4.1.4.2	time_out_secs . . . . .	8
4.1.4.3	time_out_usecs . . . . .	9
4.2	echo_server.h File Reference . . . . .	9
4.2.1	Detailed Description . . . . .	9
4.2.2	Function Documentation . . . . .	9
4.2.2.1	echo_server . . . . .	9

4.3	helper.c File Reference	10
4.3.1	Detailed Description	10
4.3.2	Macro Definition Documentation	11
4.3.2.1	MAX_BUFFER_SIZE	11
4.3.3	Function Documentation	11
4.3.3.1	print_errno_message	11
4.4	helper.h File Reference	11
4.4.1	Detailed Description	11
4.4.2	Macro Definition Documentation	12
4.4.2.1	DFPRINTF	12
4.4.2.2	DPRINTF	12
4.4.3	Function Documentation	12
4.4.3.1	print_errno_message	12
4.5	main.c File Reference	12
4.5.1	Detailed Description	13
4.5.2	Function Documentation	13
4.5.2.1	get_port_from_commandline	13
4.5.2.2	main	14
4.6	server.c File Reference	14
4.6.1	Detailed Description	15
4.6.2	Macro Definition Documentation	15
4.6.2.1	IPV6	15
4.6.3	Typedef Documentation	15
4.6.3.1	ThreadCount	15
4.6.4	Function Documentation	15
4.6.4.1	create_server_socket	15
4.6.4.2	decrement_thread_count	16
4.6.4.3	get_thread_count	16
4.6.4.4	increment_thread_count	16
4.6.4.5	start_server	16
4.6.5	Variable Documentation	16
4.6.5.1	backlog	16
4.6.5.2	thread_count	16
4.7	server.h File Reference	17
4.7.1	Detailed Description	18
4.7.2	Macro Definition Documentation	18
4.7.2.1	DDECREMENT_THREAD_COUNT	18
4.7.2.2	DINCREMENT_THREAD_COUNT	18
4.7.3	Typedef Documentation	18
4.7.3.1	ServerTag	18

4.7.4	Function Documentation	19
4.7.4.1	create_server_socket	19
4.7.4.2	decrement_thread_count	19
4.7.4.3	get_thread_count	19
4.7.4.4	increment_thread_count	19
4.7.4.5	start_server	19
4.8	socket_helpers.c File Reference	19
4.8.1	Detailed Description	20
4.8.2	Function Documentation	20
4.8.2.1	socket_readline	20
4.8.2.2	socket_readline_timeout	21
4.8.2.3	socket_writeline	21
4.9	socket_helpers.h File Reference	22
4.9.1	Detailed Description	23
4.9.2	Function Documentation	23
4.9.2.1	socket_readline	23
4.9.2.2	socket_readline_timeout	23
4.9.2.3	socket_writeline	23



# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">ServerTag</a>	
Struct for passing to server threads . . . . .	5
<a href="#">ThreadCount</a>	
Struct to synchronize access to the active thread count . . . . .	5





## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">echo_server.c</a>	Implementation of echo server functions . . . . .	7
<a href="#">echo_server.h</a>	Interface to echo server functions . . . . .	9
<a href="#">helper.c</a>	Implementation of helper functions . . . . .	10
<a href="#">helper.h</a>	Interface to helper functions . . . . .	11
<a href="#">main.c</a>	Main function for echoserver . . . . .	12
<a href="#">server.c</a>	Implementation of listening server functions . . . . .	14
<a href="#">server.h</a>	Interface to listening server functions . . . . .	17
<a href="#">socket_helpers.c</a>	Implementation of socket helper functions . . . . .	19
<a href="#">socket_helpers.h</a>	Interface to socket helper functions . . . . .	22



## Chapter 3

# Data Structure Documentation

### 3.1 ServerTag Struct Reference

Struct for passing to server threads.

```
#include <server.h>
```

#### Data Fields

- int [c\\_socket](#)

#### 3.1.1 Detailed Description

Struct for passing to server threads.

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:

- [server.h](#)

### 3.2 ThreadCount Struct Reference

Struct to synchronize access to the active thread count.

#### Data Fields

- pthread\_mutex\_t [mutex](#)
- int [count](#)

### 3.2.1 Detailed Description

Struct to synchronize access to the active thread count.

### 3.2.2 Field Documentation

#### 3.2.2.1 `int ThreadCount::count`

Active thread count variable

#### 3.2.2.2 `pthread_mutex_t ThreadCount::mutex`

Mutex for synchronized access

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

- [server.c](#)

## Chapter 4

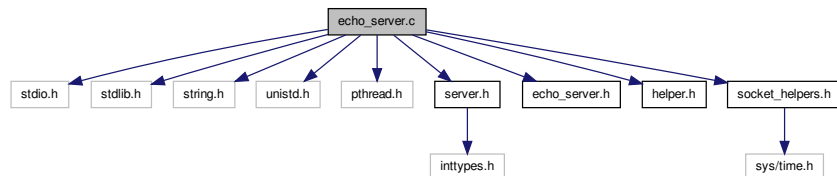
# File Documentation

### 4.1 echo\_server.c File Reference

Implementation of echo server functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include "server.h"
#include "echo_server.h"
#include "helper.h"
#include "socket_helpers.h"
```

Include dependency graph for echo\_server.c:



#### Macros

- `#define MAX_BUFFER_LEN 1024`  
*Maximum character buffer size.*

#### Functions

- `void * echo_server (void *arg)`  
*Main echo server handler thread function.*

#### Variables

- `static const long time_out_secs = 5`

*File scope variable for default time out seconds.*

- static const long `time_out_usecs` = 0

*File scope variable for default time out microseconds.*

- static const char `time_out_msg` [] = "Timeout - closing connection.\n"

*File scope variable for timeout message.*

### 4.1.1 Detailed Description

Implementation of echo server functions.

#### Author

Paul Griffiths

#### Copyright

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

### 4.1.2 Macro Definition Documentation

#### 4.1.2.1 #define MAX\_BUFFER\_LEN 1024

Maximum character buffer size.

### 4.1.3 Function Documentation

#### 4.1.3.1 void\* echo\_server ( void \* arg )

Main echo server handler thread function.

Provides echo server service to a provided connected socket. The server loops and echoes any whole lines provided. The server will time-out after a pre-defined period, if no input, or if no more input, is received.

#### Parameters

<code>arg</code>	Pointer to a <a href="#">ServerTag</a> struct
------------------	---

#### Returns

NULL

### 4.1.4 Variable Documentation

#### 4.1.4.1 const char time\_out\_msg[] = "Timeout - closing connection.\n" [static]

File scope variable for timeout message.

#### 4.1.4.2 const long time\_out\_secs = 5 [static]

File scope variable for default time out seconds.

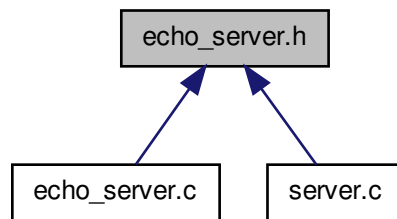
#### 4.1.4.3 `const long time_out_usecs = 0` `[static]`

File scope variable for default time out microseconds.

## 4.2 echo\_server.h File Reference

Interface to echo server functions.

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



### Functions

- `void * echo_server (void *arg)`  
Main echo server handler thread function.

#### 4.2.1 Detailed Description

Interface to echo server functions.

##### 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 `void* echo_server ( void * arg )`

Main echo server handler thread function.

Provides echo server service to a provided connected socket. The server loops and echoes any whole lines provided. The server will time-out after a pre-defined period, if no input, or if no more input, is received.

##### Parameters

<code>arg</code>	Pointer to a <a href="#">ServerTag</a> struct
------------------	---

**Returns**

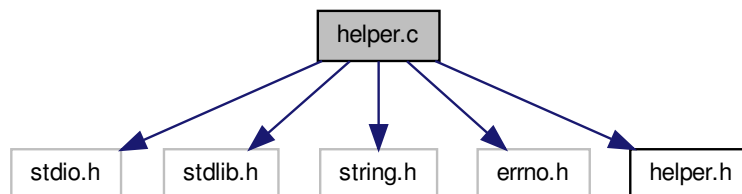
NULL

## 4.3 helper.c File Reference

Implementation of helper functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include "helper.h"
```

Include dependency graph for helper.c:

**Macros**

- #define `MAX_BUFFER_SIZE` 1024  
*Maximum character buffer size.*

**Functions**

- void `print_errno_message` (const char \*message)  
*Prints an errno error message to stderr.*

### 4.3.1 Detailed Description

Implementation of helper functions.

**Author**

Paul Griffiths

**Copyright**

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



### 4.3.2 Macro Definition Documentation

#### 4.3.2.1 #define MAX\_BUFFER\_SIZE 1024

Maximum character buffer size.

### 4.3.3 Function Documentation

#### 4.3.3.1 void print\_errno\_message ( const char \* message )

Prints an errno error message to stderr.

This function accesses the standard global variable `errno` and related function `strerror_r` (a POSIX extension) and can be called after returning from a function which sets `errno` and returns with an error code.

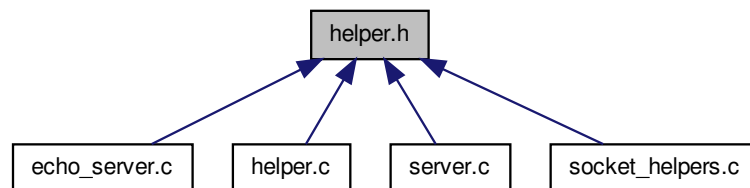
##### Parameters

<i>message</i>	The error message to show.
----------------	----------------------------

## 4.4 helper.h File Reference

Interface to helper functions.

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



### Macros

- #define `DPRINTF`(arg) printf arg  
Calls `printf()` only when `DEBUG` is defined.
- #define `DFPRINTF`(arg) fprintf arg  
Calls `fprintf()` only when `DEBUG` is defined.

### Functions

- void `print_errno_message` (const char \*message)  
Prints an errno error message to stderr.

#### 4.4.1 Detailed Description

Interface to helper functions. Interface to helper functions.

**Author**

Paul Griffiths

**Copyright**

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

**4.4.2 Macro Definition Documentation****4.4.2.1 #define DFPRINTF( *arg* ) fprintf arg**

Calls fprintf() only when DEBUG is defined.

**Parameters**

<i>arg</i>	The normal parameters to fprintf()
------------	------------------------------------

**4.4.2.2 #define DPRINTF( *arg* ) printf arg**

Calls printf() only when DEBUG is defined.

**Parameters**

<i>arg</i>	The normal parameters to printf()
------------	-----------------------------------

**4.4.3 Function Documentation****4.4.3.1 void print\_errno\_message ( const char \* *message* )**

Prints an errno error message to stderr.

This function accesses the standard global variable `errno` and related function `strerror_r` (a POSIX extension) and can be called after returning from a function which sets `errno` and returns with an error code.

**Parameters**

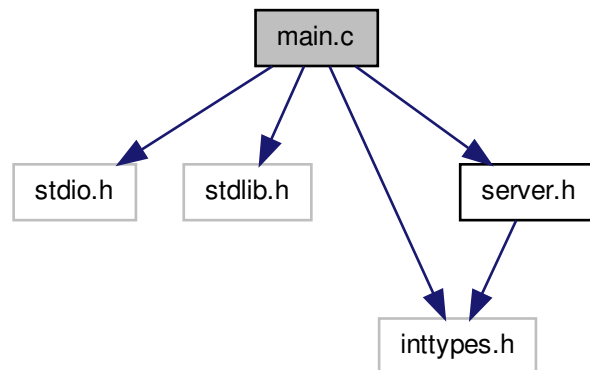
<i>message</i>	The error message to show.
----------------	----------------------------

**4.5 main.c File Reference**

Main function for echoserver.

```
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "server.h"
```

Include dependency graph for main.c:



## Functions

- `uint16_t get_port_from_commandline (const int argc, char **argv)`  
Parses the command line for a specified TCP port.
- `int main (int argc, char **argv)`  
Main function.

### 4.5.1 Detailed Description

Main function for echoserver.

#### Author

Paul Griffiths

#### Copyright

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

### 4.5.2 Function Documentation

#### 4.5.2.1 `uint16_t get_port_from_commandline ( const int argc, char ** argv )`

Parses the command line for a specified TCP port.

Checks for the existence of a single command line argument, and if one and only one is present, attempts to interpret it as a TCP listening port, between 1 and 49151 (ports above 49151 are ephemeral ports).

#### Parameters

<code>argc</code>	The number of command line arguments, passed from <code>main()</code>
<code>argv</code>	The command line arguments, passed from <code>main()</code>

**Returns**

The specified TCP port if successful, or 0 on error.

**4.5.2.2 int main ( int argc, char \*\* argv )**

Main function.

Main function.

**Returns**

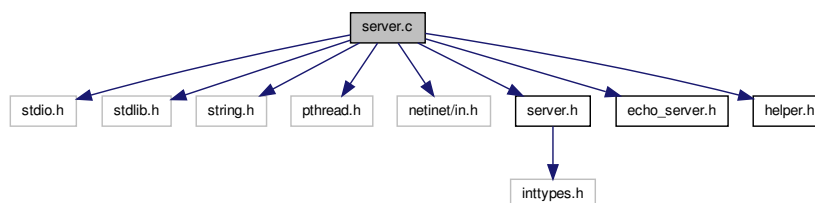
Exit status.

**4.6 server.c File Reference**

Implementation of listening server functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <netinet/in.h>
#include "server.h"
#include "echo_server.h"
#include "helper.h"
```

Include dependency graph for server.c:

**Data Structures**

- struct [ThreadCount](#)  
*Struct to synchronize access to the active thread count.*

**Macros**

- #define [IPV6](#)  
*Create an IPv6 rather than IPv4 listening socket.*

**Typedefs**

- typedef struct [ThreadCount](#) ThreadCount  
*Struct to synchronize access to the active thread count.*

## Functions

- int `create_server_socket` (const uint16\_t listening\_port)  
*Creates a listening socket.*
- int `start_server` (const int listening\_socket)  
*Starts an active server.*
- int `get_thread_count` (void)  
*Gets the active thread count.*
- void `increment_thread_count` (void)  
*Increments the active thread count.*
- void `decrement_thread_count` (void)  
*Decrements the active thread count.*

## Variables

- static const int `backlog` = 1024  
*File scope variable for default backlog.*
- static `ThreadCount thread_count` = {PTHREAD\_MUTEX\_INITIALIZER, 0}  
*File scope variable holding the active thread count.*

### 4.6.1 Detailed Description

Implementation of listening server functions.

#### Author

Paul Griffiths

#### Copyright

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

### 4.6.2 Macro Definition Documentation

#### 4.6.2.1 #define IPV6

Create an IPv6 rather than IPv4 listening socket.

### 4.6.3 Typedef Documentation

#### 4.6.3.1 typedef struct ThreadCount ThreadCount

Struct to synchronize access to the active thread count.

### 4.6.4 Function Documentation

#### 4.6.4.1 int create\_server\_socket ( const uint16\_t listening\_port )

Creates a listening socket.

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

**Parameters**

<i>listening_port</i>	The port the socket should listen on
-----------------------	--------------------------------------

**Returns**

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

**4.6.4.2 void decrement\_thread\_count ( void )**

Decrements the active thread count.

Used for debugging purposes to check that threads are exiting and being destroyed when expected.

**4.6.4.3 int get\_thread\_count ( void )**

Gets the active thread count.

Used for debugging purposes to check that threads are exiting and being destroyed when expected.

**Returns**

The number of active threads (excluding the main thread).

**4.6.4.4 void increment\_thread\_count ( void )**

Increments the active thread count.

Used for debugging purposes to check that threads are exiting and being destroyed when expected.

**4.6.4.5 int start\_server ( const int listening\_socket )**

Starts an active server.

Connections are passed to a new server thread.

**Parameters**

<i>listening_socket</i>	A file descriptor for a listening socket.
-------------------------	---

**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.

**4.6.5 Variable Documentation****4.6.5.1 const int backlog = 1024 [static]**

File scope variable for default backlog.

Determines the maximum length to which the queue of pending connections may grow. Used when calling listen().

**4.6.5.2 ThreadCount thread\_count = {PTHREAD\_MUTEX\_INITIALIZER, 0} [static]**

File scope variable holding the active thread count.

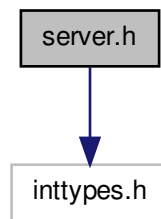
This variable is accessed and manipulated solely through the [get\\_thread\\_count\(\)](#), [increment\\_thread\\_count\(\)](#), and [decrement\\_thread\\_count\(\)](#) functions.

## 4.7 server.h File Reference

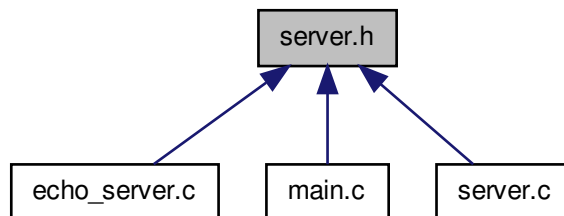
Interface to listening server functions.

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

Include dependency graph for server.h:



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



### Data Structures

- struct [ServerTag](#)  
*Struct for passing to server threads.*

### Macros

- #define [DINCREMENT\\_THREAD\\_COUNT](#)(arg) [increment\\_thread\\_count](#)()  
*Calls [increment\\_thread\\_count\(\)](#) only if `DEBUG` is defined.*
- #define [DDECREMENT\\_THREAD\\_COUNT](#)(arg) [decrement\\_thread\\_count](#)()  
*Calls [decrement\\_thread\\_count\(\)](#) only if `DEBUG` is defined.*

## Typedefs

- typedef struct [ServerTag](#) [ServerTag](#)

*Struct for passing to server threads.*

## Functions

- int [create\\_server\\_socket](#) (const uint16\_t listening\_port)  
*Creates a listening socket.*
- int [start\\_server](#) (const int listening\_socket)  
*Starts an active server.*
- void [increment\\_thread\\_count](#) (void)  
*Increments the active thread count.*
- void [decrement\\_thread\\_count](#) (void)  
*Decrements the active thread count.*
- int [get\\_thread\\_count](#) (void)  
*Gets the active thread count.*

### 4.7.1 Detailed Description

Interface to listening server functions.

#### Author

Paul Griffiths

#### Copyright

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

### 4.7.2 Macro Definition Documentation

#### 4.7.2.1 #define DDECREMENT\_THREAD\_COUNT( *arg* ) [decrement\\_thread\\_count\(\)](#)

Calls [decrement\\_thread\\_count\(\)](#) only if DEBUG is defined.

#### 4.7.2.2 #define DINCREMENT\_THREAD\_COUNT( *arg* ) [increment\\_thread\\_count\(\)](#)

Calls [increment\\_thread\\_count\(\)](#) only if DEBUG is defined.

### 4.7.3 Typedef Documentation

#### 4.7.3.1 typedef struct [ServerTag](#) [ServerTag](#)

Struct for passing to server threads.

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



## 4.7.4 Function Documentation

### 4.7.4.1 int create\_server\_socket ( const uint16\_t *listening\_port* )

Creates a listening socket.

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

#### Parameters

<i>listening_port</i>	The port the socket should listen on
-----------------------	--------------------------------------

#### Returns

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

### 4.7.4.2 void decrement\_thread\_count ( void )

Decrements the active thread count.

Used for debugging purposes to check that threads are exiting and being destroyed when expected.

### 4.7.4.3 int get\_thread\_count ( void )

Gets the active thread count.

Used for debugging purposes to check that threads are exiting and being destroyed when expected.

#### Returns

The number of active threads (excluding the main thread).

### 4.7.4.4 void increment\_thread\_count ( void )

Increments the active thread count.

Used for debugging purposes to check that threads are exiting and being destroyed when expected.

### 4.7.4.5 int start\_server ( const int *listening\_socket* )

Starts an active server.

Connections are passed to a new server thread.

#### Parameters

<i>listening_socket</i>	A file descriptor for a listening socket.
-------------------------	---

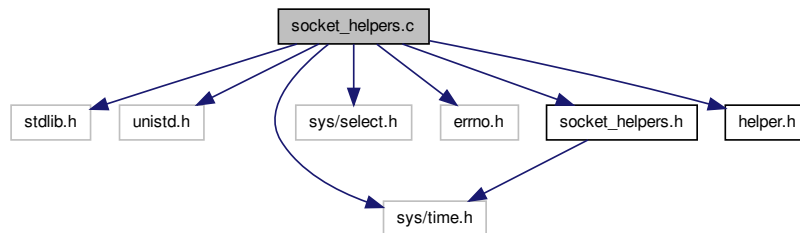
#### Returns

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.

## 4.8 socket\_helpers.c File Reference

Implementation of socket helper functions.

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/select.h>
#include <errno.h>
#include "socket_helpers.h"
#include "helper.h"
Include dependency graph for socket_helpers.c:
```



## Functions

- ssize\_t [socket\\_readline](#) (const int socket, char \*buffer, const size\_t max\_len)  
*Reads a \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 a \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.*

### 4.8.1 Detailed Description

Implementation of socket helper functions. Implementation of socket helper functions.

#### Author

Paul Griffiths

#### Copyright

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

### 4.8.2 Function Documentation

#### 4.8.2.1 ssize\_t socket\_readline ( const int socket, char \* buffer, const size\_t max\_len )

Reads a \n terminated line from a socket.

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.

## Parameters

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

## Returns

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

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

Reads a \n terminated line from a socket with timeout.

Behaves the same as [socket\\_readline\(\)](#), except it will time out if no input is available on the socket after the specified time.

## Parameters

<i>socket</i>	File description of the socket
<i>buffer</i>	The buffer into which to read
<i>max_len</i>	The maximum number of characters to read, including the terminating \0.
<i>time_out</i>	A pointer to a <code>timeval</code> struct containing the timeout period. Note that some implementations of <code>select()</code> 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.

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

Writes a line to a socket.

## Parameters

<i>socket</i>	File description of the socket
<i>buffer</i>	The buffer from which to write.
<i>max_len</i>	The maximum number of characters to read from the buffer.

### Returns

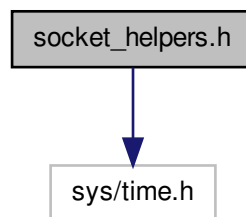
The number of characters written, or -1 on encountering an error.

## 4.9 socket\_helpers.h File Reference

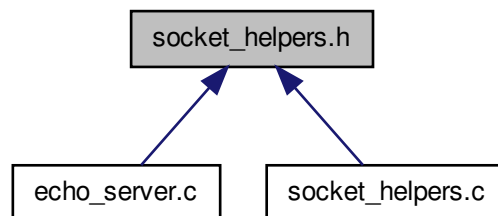
Interface to socket helper functions.

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

Include dependency graph for socket\_helpers.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 a \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 a \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.*

### 4.9.1 Detailed Description

Interface to socket helper functions.

#### Author

Paul Griffiths

#### Copyright

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

### 4.9.2 Function Documentation

#### 4.9.2.1 `ssize_t socket_readline ( const int socket, char * buffer, const size_t max_len )`

Reads a `\n` terminated line from a socket.

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`.

##### Parameters

<i>socket</i>	File description of the socket
<i>buffer</i>	The buffer into which to read
<i>max_len</i>	The maximum number of characters to read, including the terminating <code>\0</code> .

##### Returns

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

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

Reads a `\n` terminated line from a socket with timeout.

Behaves the same as [socket\\_readline\(\)](#), except it will time out if no input is available on the socket after the specified time.

##### Parameters

<i>socket</i>	File description of the socket
<i>buffer</i>	The buffer into which to read
<i>max_len</i>	The maximum number of characters to read, including the terminating <code>\0</code> .
<i>time_out</i>	A pointer to a <code>timeval</code> struct containing the timeout period. Note that some implementations of <code>select()</code> 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.

#### 4.9.2.3 `ssize_t socket_writeline ( const int socket, const char * buffer, const size_t max_len )`

Writes a line to a socket.

**Parameters**

<i>socket</i>	File description of the socket
<i>buffer</i>	The buffer from which to write.
<i>max_len</i>	The maximum number of characters to read from the buffer.

**Returns**

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

# Index

- backlog
  - server.c, 16
- c\_socket
  - ServerTag, 5
- count
  - ThreadCount, 6
- create\_server\_socket
  - server.c, 15
  - server.h, 19
- DFPRINTF
  - helper.h, 12
- DPRINTF
  - helper.h, 12
- decrement\_thread\_count
  - server.c, 16
  - server.h, 19
- echo\_server
  - echo\_server.c, 8
  - echo\_server.h, 9
- echo\_server.c, 7
  - echo\_server, 8
  - MAX\_BUFFER\_LEN, 8
  - time\_out\_msg, 8
  - time\_out\_secs, 8
  - time\_out\_usecs, 8
- echo\_server.h, 9
  - echo\_server, 9
- get\_port\_from\_commandline
  - main.c, 13
- get\_thread\_count
  - server.c, 16
  - server.h, 19
- helper.c, 10
  - MAX\_BUFFER\_SIZE, 11
  - print\_errno\_message, 11
- helper.h, 11
  - DFPRINTF, 12
  - DPRINTF, 12
  - print\_errno\_message, 12
- IPV6
  - server.c, 15
- increment\_thread\_count
  - server.c, 16
  - server.h, 19
- MAX\_BUFFER\_LEN
  - echo\_server.c, 8
- MAX\_BUFFER\_SIZE
  - helper.c, 11
- main
  - main.c, 14
- main.c, 12
  - get\_port\_from\_commandline, 13
  - main, 14
- mutex
  - ThreadCount, 6
- print\_errno\_message
  - helper.c, 11
  - helper.h, 12
- server.c, 14
  - backlog, 16
  - create\_server\_socket, 15
  - decrement\_thread\_count, 16
  - get\_thread\_count, 16
  - IPV6, 15
  - increment\_thread\_count, 16
  - start\_server, 16
  - thread\_count, 16
  - ThreadCount, 15
- server.h, 17
  - create\_server\_socket, 19
  - decrement\_thread\_count, 19
  - get\_thread\_count, 19
  - increment\_thread\_count, 19
  - ServerTag, 18
  - start\_server, 19
- ServerTag, 5
  - c\_socket, 5
  - server.h, 18
- socket\_helpers.c, 19
  - socket\_readline, 20
  - socket\_readline\_timeout, 21
  - socket\_writeline, 21
- socket\_helpers.h, 22
  - socket\_readline, 23
  - socket\_readline\_timeout, 23
  - socket\_writeline, 23
- socket\_readline
  - socket\_helpers.c, 20
  - socket\_helpers.h, 23
- socket\_readline\_timeout
  - socket\_helpers.c, 21
  - socket\_helpers.h, 23

- socket\_writeline
  - socket\_helpers.c, [21](#)
  - socket\_helpers.h, [23](#)
- start\_server
  - server.c, [16](#)
  - server.h, [19](#)
- thread\_count
  - server.c, [16](#)
- ThreadCount, [5](#)
  - count, [6](#)
  - mutex, [6](#)
  - server.c, [15](#)
- time\_out\_msg
  - echo\_server.c, [8](#)
- time\_out\_secs
  - echo\_server.c, [8](#)
- time\_out\_usecs
  - echo\_server.c, [8](#)