echoserver

Generated by Doxygen 1.8.1.2

Fri Aug 30 2013 19:58:09

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 files with brief descriptions:

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

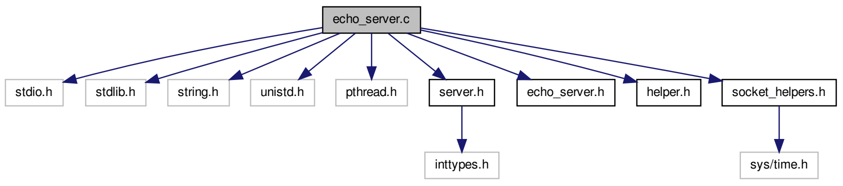
File Documentation

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

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

arg	Pointer to a ServerTag 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.

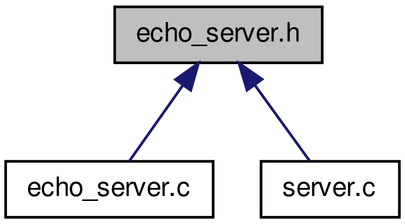
4.1.4.3 const long time_out_usecs = 0 `[static]`

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

arg	Pointer to a ServerTag struct

Returns

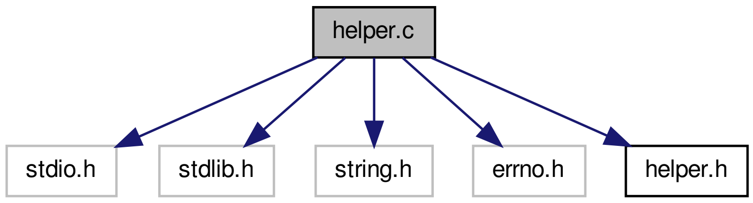
NULL

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

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.

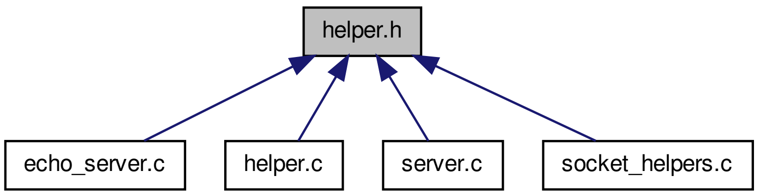
Parameters

message	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

arg	The normal parameters to fprintf()

4.4.2.2 #define DPRINTF(*arg*) printf arg

Calls printf() only when DEBUG is defined.

Parameters

arg	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

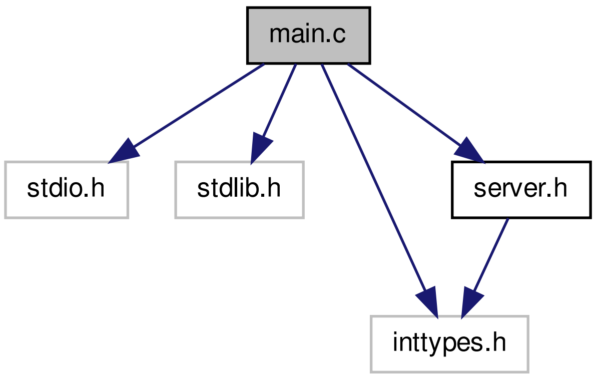
message	The error message to show.

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

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

argc	The number of command line arguments, passed from main()
argv	The command line arguments, passed from main()

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

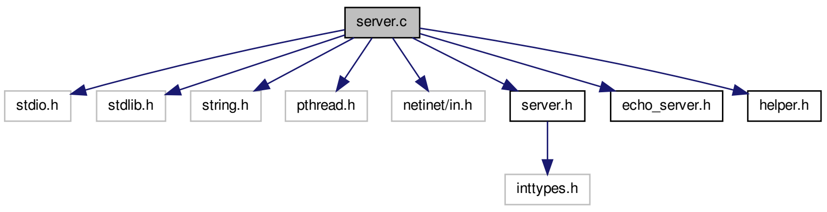
Exit status.

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

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.

4.6.4.2 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

listening_socket	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.

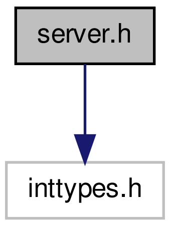
This variable is accessed and manipulated solely through the get_thread_count(), increment_thread_count(), and decrement_thread_count() functions.

4.7 server.h File Reference

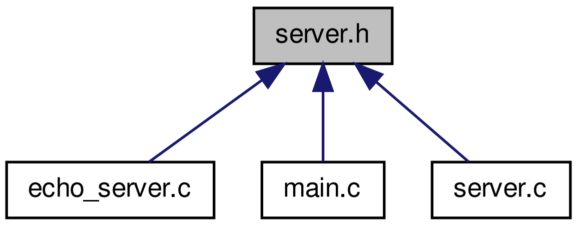
Interface to listening server functions.

`#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

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.

4.7.4.2 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

listening_socket	A file descriptor for a listening socket.

Returns

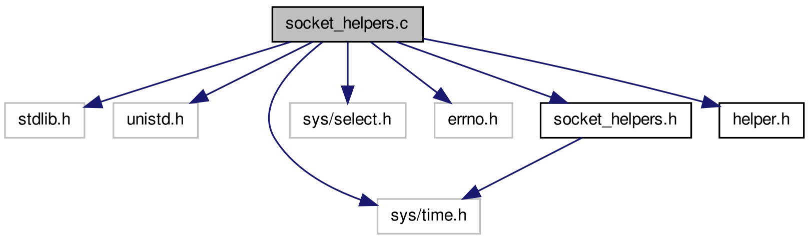
Returns non-zero on encountering an error. The server runs in an infinite loop, and this function will not return unless an error is countered.

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

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.

4.8.2.2 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

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.

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

Writes a line to a socket.

Parameters

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

Returns

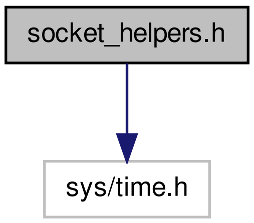
The number of characters written, or -1 on encountering an error.

4.9 socket_helpers.h File Reference

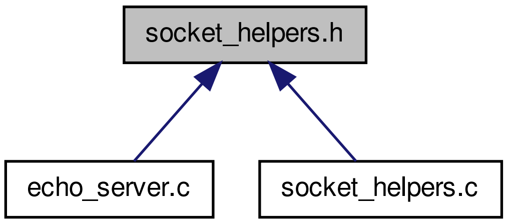
Interface to socket helper functions.

`#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

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.

4.9.2.2 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

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.

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

Writes a line to a socket.

Parameters

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

Returns

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

Index