ZipZop

Generated by Doxygen 1.8.13

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 client Struct Reference

Struct representing a connect client in the server.

Data Fields

- const char ∗ name
- int sockfd
- pthread_t thread

3.1.1 Detailed Description

Struct representing a connect client in the server.

3.1.2 Field Documentation

3.1.2.1 name

```
const char* client::name
```

Client name

3.1.2.2 sockfd

```
int client::sockfd
```

Socket that holds the connection with this client

3.1.2.3 thread

```
pthread_t client::thread
```

The server thread responsible to listen to this client's messages

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

- src/client.c

3.2 message Struct Reference

Struct representing a messege sent by some sender.

Data Fields

- const char * content
- const char * sender_name

3.2.1 Detailed Description

Struct representing a messege sent by some sender.

3.2.2 Field Documentation

3.2.2.1 content

```
const char* message::content
```

The content of the message

3.2.2.2 sender_name

```
const char* message::sender_name
```

The username of the sender

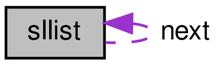
The documentation for this struct was generated from the following file:

- src/message.c

3.3 sllist Struct Reference

A struct representing node in a singly linked list.

Collaboration diagram for sllist:



Data Fields

- void ∗ key
- struct sllist ∗ next

3.3.1 Detailed Description

A struct representing node in a singly linked list.

3.3.2 Field Documentation

3.3.2.1 key

```
void* sllist::key
```

The element that will be stored in the node

3.3.2.2 next

```
struct sllist* sllist::next
```

A pointer to the next node

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

- src/sllist.c

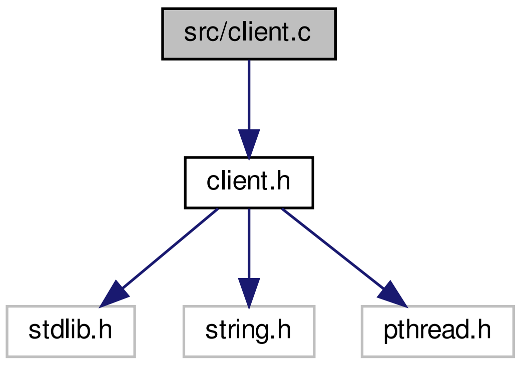
Chapter 4

File Documentation

4.1 src/client.c File Reference

```
#include "client.h"
```

Include dependency graph for client.c:



Data Structures

- struct client

  Struct representing a connect client in the server.

Functions

- struct client ∗ client_create (const char ∗name, int sockfd)

  Create a client instance.
- void client_destroy (struct client ∗c)
- const char ∗ client_get_name (struct client ∗c)

Get the client name.
- int client_get_socket (struct client ∗c)

  Get the client socket.
- pthread_t ∗ client_get_thread (struct client ∗c)

  Get the client thread.
- void client_set_name (struct client ∗c, const char ∗name)

  Set the client name.
- void client_set_socket (struct client ∗c, int sockfd)

  Set the client socket.
- void client_set_thread (struct client ∗c, pthread_t thread)

  Set the client thread.

4.1.1 Function Documentation

4.1.1.1 client_create()

```
struct client* client_create (
              const char * name,
              int sockfd )
```

Create a client instance.

Both parameters will be copied into the message, so the user is free to `free()` the parameters passed to this function if necessary.

Parameters

in	*name*	The client name.
in	*sockfd*	The socket connected to this client.

Returns

A pointer to the client in case of success, NULL otherwise. The client must be freed, using client_destroy().

See also

client_destroy

4.1.1.2 client_destroy()

```
void client_destroy (
              struct client * c )
```

Destroys a client.

Parameters

in	*c*	A pointer to the client.

4.1.1.3 client_get_name()

```
const char* client_get_name (
             struct client * c )
```

Get the client name.

Parameters

in	*c*	The client.

Returns

The client name.

4.1.1.4 client_get_socket()

```
int client_get_socket (
             struct client * c )
```

Get the client socket.

Parameters

in	*c*	The client.

Returns

The client socket.

4.1.1.5 client_get_thread()

```
pthread_t* client_get_thread (
             struct client * c )
```

Get the client thread.

Parameters

in	*c*	The client.

Returns

An Address of the client thread.

Warning

This function returns the address of the actual thread stored in the client. Do not try to free this address.

4.1.1.6 client_set_name()

```
void client_set_name (
             struct client * c,
             const char * name )
```

Set the client name.

Parameters

in	*c*	The client.
in	*name*	The client name.

4.1.1.7 client_set_socket()

```
void client_set_socket (
             struct client * c,
             int sockfd )
```

Set the client socket.

Parameters

in	*c*	The client.
in	*sockfd*	The client socket.

4.1.1.8 client_set_thread()

```
void client_set_thread (
             struct client * c,
             pthread_t thread )
```

Set the client thread.

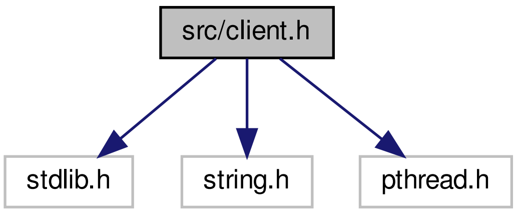
Parameters

in	*c*	The client.
in	*thread*	The client thread.

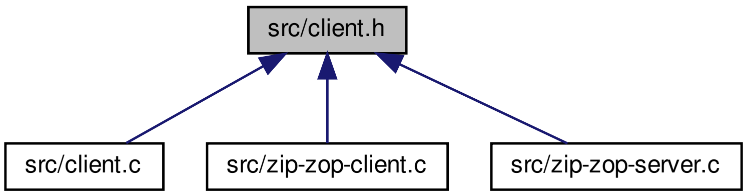
4.2 src/client.h File Reference

```
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
```

Include dependency graph for client.h:



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



Functions

- struct client ∗ client_create (const char ∗name, int sockfd)

  Create a client instance.
- void client_destroy (struct client ∗c)
- const char ∗ client_get_name (struct client ∗c)

  Get the client name.

- int client_get_socket (struct client ∗c)

    Get the client socket.
- pthread_t ∗ client_get_thread (struct client ∗c)

    Get the client thread.
- void client_set_name (struct client ∗c, const char ∗name)

    Set the client name.
- void client_set_socket (struct client ∗c, int sockfd)

    Set the client socket.
- void client_set_thread (struct client ∗c, pthread_t thread)

    Set the client thread.

4.2.1 Function Documentation

4.2.1.1 client_create()

```
struct client* client_create (
              const char * name,
              int sockfd )
```

Create a client instance.

Both parameters will be copied into the message, so the user is free to `free()` the parameters passed to this function if necessary.

Parameters

in	*name*	The client name.
in	*sockfd*	The socket connected to this client.

Returns

A pointer to the client in case of success, NULL otherwise. The client must be freed, using client_destroy().

See also

client_destroy

4.2.1.2 client_destroy()

```
void client_destroy (
              struct client * c )
```

Destroys a client.

Parameters

in	*c*	A pointer to the client.

4.2.1.3 client_get_name()

```
const char* client_get_name (
             struct client * c )
```

Get the client name.

Parameters

in	*c*	The client.

Returns

The client name.

4.2.1.4 client_get_socket()

```
int client_get_socket (
             struct client * c )
```

Get the client socket.

Parameters

in	*c*	The client.

Returns

The client socket.

4.2.1.5 client_get_thread()

```
pthread_t* client_get_thread (
             struct client * c )
```

Get the client thread.

Parameters

in	*c*	The client.

Returns

An Address of the client thread.

Warning

This function returns the address of the actual thread stored in the client. Do not try to free this address.

4.2.1.6 client_set_name()

```
void client_set_name (
             struct client * c,
             const char * name )
```

Set the client name.

Parameters

in	*c*	The client.
in	*name*	The client name.

4.2.1.7 client_set_socket()

```
void client_set_socket (
             struct client * c,
             int sockfd )
```

Set the client socket.

Parameters

in	*c*	The client.
in	*sockfd*	The client socket.

4.2.1.8 client_set_thread()

```
void client_set_thread (
             struct client * c,
             pthread_t thread )
```

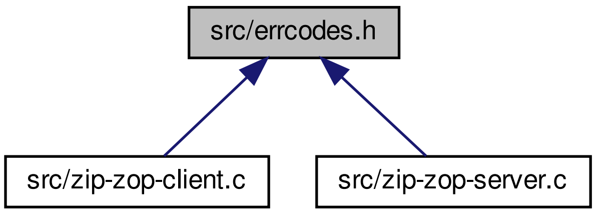
Set the client thread.

Parameters

in	*c*	The client.
in	*thread*	The client thread.

4.3 src/errcodes.h File Reference

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



Enumerations

- enum errcodes {
  E_SUCCESS, E_GETADDRINFO, E_BIND, E_LISTEN,
  E_BAD_ARGS, E_CONNECT, E_PTHREAD_CREATE }

    Possible error codes in the project.

4.3.1 Enumeration Type Documentation

4.3.1.1 errcodes

```
enum errcodes
```

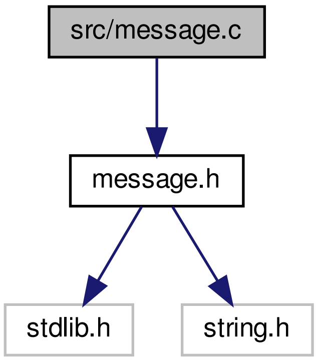
Possible error codes in the project.

Enumerator

E_SUCCESS	Success value
E_GETADDRINFO	Error code if getaddrinfo() fails
E_BIND	Error code if it was not possible to bind() in the specified port
E_LISTEN	Error code if liste() fails
E_BAD_ARGS	Error code if the user gave a bad input
E_CONNECT	Error code if connect() fails
E_PTHREAD_CREATE	Error code if it was not possible to create a new thread

4.4 src/message.c File Reference

```
#include "message.h"
```
Include dependency graph for message.c:



Data Structures

- struct message

  Struct representing a messege sent by some sender.

Functions

- struct message ∗ message_create (const char ∗content, const char ∗sender_name)

  Creates a message.
- void message_destroy (struct message ∗m)

  Destroys a message.
- const char ∗ message_get_content (struct message ∗m)

  Get the message content.
- const char ∗ message_get_sender (struct message ∗m)

  Get the message sender.
- char ∗ message_pack (struct message ∗m, int ∗len)

  Serialize a message.
- struct message ∗ message_unpack (char ∗pack)

4.4.1 Function Documentation

4.4.1.1 message_create()

```
struct message* message_create (
            const char * content,
            const char * sender_name )
```

Creates a message.

Both parameters will be copied into the message, so the user is free to `free()` the parameters passed to this function if necessary.

Parameters

in	*content*	The content of the message.
in	*sender_name*	The username of the sender.

Returns

A pointer to a struct message in case of success, NULL otherwise. The message must be freed, using message_destroy(), when is not needed anymore.

See also

message_destroy

4.4.1.2 message_destroy()

```
void message_destroy (
            struct message * m )
```

Destroys a message.

Parameters

in	*m*	A pointer to the message.

See also

message_create

4.4.1.3 message_get_content()

```
const char* message_get_content (
            struct message * m )
```

Get the message content.

Parameters

in	*m*	A pointer to the message.

Returns

A pointer to the message content.

Warning

The returned value should not be freed.

4.4.1.4 message_get_sender()

```
const char* message_get_sender (
            struct message * m )
```

Get the message sender.

Parameters

in	*m*	A pointer to the message.

Returns

A pointer to the sender name.

Warning

The returned value should not be freed.

4.4.1.5 message_pack()

```
char* message_pack (
            struct message * m,
            int * len )
```

Serialize a message.

Pack/Serialize the struct message in a format that can be sent through the network.

Parameters

in	*m*	A pointer to the message.
out	*len*	A pointer to a integer where the length of the serialized message will be stored.

Returns

A pointer to the serialized message. This should be freed when is not necessary anymore.

See also

message_unpack

4.4.1.6 message_unpack()

```
struct message* message_unpack (
            char * pack )
```

Deserialize a message.

Unpack/Deserialize a string into a struct message.

Parameters

in	*pack*	The string that represent the packed message generated by message_pack().

Returns

A pointer to the deserialized message. This should be freed when is not necessary anymore.

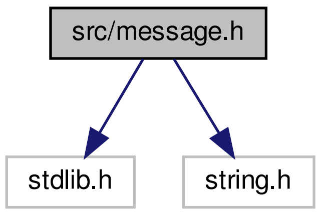
See also

message_pack

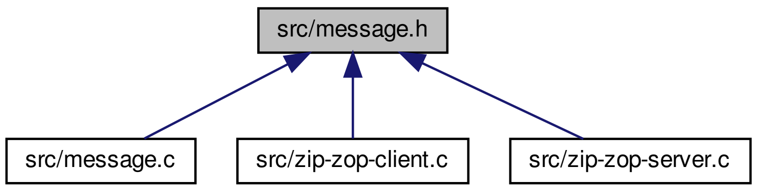
4.5 src/message.h File Reference

```
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for message.h:



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



Functions

- struct message ∗ message_create (const char ∗content, const char ∗sender_name)

    Creates a message.
- void message_destroy (struct message ∗m)

    Destroys a message.
- const char ∗ message_get_content (struct message ∗m)

    Get the message content.
- const char ∗ message_get_sender (struct message ∗m)

    Get the message sender.
- char ∗ message_pack (struct message ∗m, int ∗len)

    Serialize a message.
- struct message ∗ message_unpack (char ∗pack)

4.5.1 Function Documentation

4.5.1.1 message_create()

```
struct message* message_create (
            const char * content,
            const char * sender_name )
```

Creates a message.

Both parameters will be copied into the message, so the user is free to `free()` the parameters passed to this function if necessary.

Parameters

in	*content*	The content of the message.
in	*sender_name*	The username of the sender.

Returns

A pointer to a struct message in case of success, NULL otherwise. The message must be freed, using message_destroy(), when is not needed anymore.

See also

message_destroy

4.5.1.2 message_destroy()

```
void message_destroy (
             struct message * m )
```

Destroys a message.

Parameters

in	*m*	A pointer to the message.

See also

message_create

4.5.1.3 message_get_content()

```
const char* message_get_content (
             struct message * m )
```

Get the message content.

Parameters

in	*m*	A pointer to the message.

Returns

A pointer to the message content.

Warning

The returned value should not be freed.

4.5.1.4 message_get_sender()

```
const char* message_get_sender (
            struct message * m )
```

Get the message sender.

Parameters

in	*m*	A pointer to the message.

Returns

A pointer to the sender name.

Warning

The returned value should not be freed.

4.5.1.5 message_pack()

```
char* message_pack (
            struct message * m,
            int * len )
```

Serialize a message.

Pack/Serialize the struct message in a format that can be sent through the network.

Parameters

in	*m*	A pointer to the message.
out	*len*	A pointer to a integer where the length of the serialized message will be stored.

Returns

A pointer to the serialized message. This should be freed when is not necessary anymore.

See also

message_unpack

4.5.1.6 message_unpack()

```
struct message* message_unpack (
             char * pack )
```

Deserialize a message.

Unpack/Deserialize a string into a struct message.

Parameters

in	*pack*	The string that represent the packed message generated by message_pack().

Returns

A pointer to the deserialized message. This should be freed when is not necessary anymore.

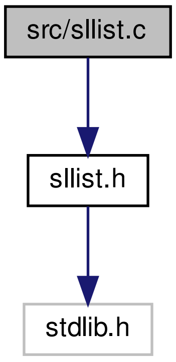
See also

message_pack

4.6 src/sllist.c File Reference

```
#include "sllist.h"
```

Include dependency graph for sllist.c:



Data Structures

- struct sllist

  A struct representing node in a singly linked list.

Functions

- struct sllist ∗ sll_init (void)

  Initilize a sllist to be a valid empty list.
- struct sllist ∗ sll_get_next (struct sllist ∗∗l)

  Get the next node in the list.
- void sll_insert_first (struct sllist ∗∗l, void ∗a)

  Insert an element on the head of the list.
- void sll_insert_last (struct sllist ∗∗l, void ∗a)

  Insert an element on the tail of the list.
- void ∗ sll_remove_first (struct sllist ∗∗l)

  Remove the first element of the list.
- void ∗ sll_remove_last (struct sllist ∗∗l)

  Remove the last element of the list.
- void ∗ sll_remove_elm (struct sllist ∗∗l, void ∗elm)

  Remove the specified element of the list.
- void ∗ sll_get_key (struct sllist ∗l)

  Get the element stored in the especified list node.

4.6.1 Function Documentation

4.6.1.1 sll_get_key()

```
void* sll_get_key (
            struct sllist * l )
```

Get the element stored in the especified list node.

Parameters

in	*l*	A pointer to the list node.

Returns

The element.

4.6.1.2 sll_get_next()

```
struct sllist* sll_get_next (
            struct sllist ** l )
```

Get the next node in the list.

Parameters

in,out	*l*	An address to a pointer to the list.

Returns

A pointer to the next node in the list; NULL if there is no next element.

Example to interate over a list:

```
struct sllist *l = sll_init();
// fill the list
for (struct sllist *p = l; p; p = sll_get_next(&p)) {
    void *key = sll_get_key(p);
    // do stuff with p
}
```

4.6.1.3 sll_init()

```
struct sllist* sll_init (
            void  )
```

Initilize a sllist to be a valid empty list.

Returns

An empty list.

Warning

One should not test the return against NULL. NULL is the default value.

See also

SLL_INIT

4.6.1.4 sll_insert_first()

```
void sll_insert_first (
            struct sllist ** l,
            void * a )
```

Insert an element on the head of the list.

Parameters

in,out	*l*	An address to a pointer to the list.
in	*a*	The element.

4.6.1.5 sll_insert_last()

```
void sll_insert_last (
             struct sllist ** l,
             void * a )
```

Insert an element on the tail of the list.

Parameters

in,out	*l*	An address to a pointer to the list.
in	*a*	The element.

4.6.1.6 sll_remove_elm()

```
void* sll_remove_elm (
             struct sllist ** l,
             void * elm )
```

Remove the specified element of the list.

Parameters

in,out	*l*	An address to a pointer to the list.
in	*elm*	The element.

Returns

The element in case of success. NULL if the list is empty or the element doesn't exit.

4.6.1.7 sll_remove_first()

```
void* sll_remove_first (
             struct sllist ** l )
```

Remove the first element of the list.

The list node will be freed.

Parameters

`in,out`	*l*	An address to a pointer to the list.

Returns

The element in case of success. NULL if the list is empty.

4.6.1.8 sll_remove_last()

```
void* sll_remove_last (
            struct sllist ** l )
```

Remove the last element of the list.

The list node will be freed.

Parameters

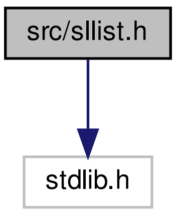
`in,out`	*l*	An address to a pointer to the list.

Returns

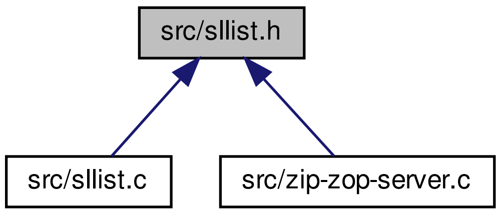
The element in case of success. NULL if the list is empty.

4.7 src/sllist.h File Reference

`#include <stdlib.h>`
Include dependency graph for sllist.h:



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



Macros

- #define SLL_INIT() NULL;

  Macro that initialize a sllist to be a valid empty list.

Functions

- struct sllist ∗ sll_init (void)

  Initilize a sllist to be a valid empty list.
- struct sllist ∗ sll_get_next (struct sllist ∗∗l)

  Get the next node in the list.
- void sll_insert_first (struct sllist ∗∗l, void ∗a)

  Insert an element on the head of the list.
- void sll_insert_last (struct sllist ∗∗l, void ∗a)

  Insert an element on the tail of the list.
- void ∗ sll_remove_first (struct sllist ∗∗l)

  Remove the first element of the list.
- void ∗ sll_remove_last (struct sllist ∗∗l)

  Remove the last element of the list.
- void ∗ sll_remove_elm (struct sllist ∗∗l, void ∗elm)

  Remove the specified element of the list.
- void ∗ sll_get_key (struct sllist ∗l)

  Get the element stored in the especified list node.

4.7.1 Macro Definition Documentation

4.7.1.1 SLL_INIT

```
#define SLL_INIT( ) NULL;
```

Macro that initialize a sllist to be a valid empty list.

Returns

An empty list.

Warning

One should not test the return against NULL. NULL is the default value.

See also

sll_init

4.7.2 Function Documentation

4.7.2.1 sll_get_key()

```
void* sll_get_key (
              struct sllist * l )
```

Get the element stored in the especified list node.

Parameters

in	*l*	A pointer to the list node.

Returns

The element.

4.7.2.2 sll_get_next()

```
struct sllist* sll_get_next (
              struct sllist ** l )
```

Get the next node in the list.

Parameters

in,out	*l*	An address to a pointer to the list.

Returns

A pointer to the next node in the list; NULL if there is no next element.

Example to interate over a list:

```
struct sllist *l = sll_init();
// fill the list
for (struct sllist *p = l; p; p = sll_get_next(&p)) {
    void *key = sll_get_key(p);
    // do stuff with p
}
```

4.7.2.3 sll_init()

```
struct sllist* sll_init (
              void  )
```

Initilize a sllist to be a valid empty list.

Returns

An empty list.

Warning

One should not test the return against NULL. NULL is the default value.

See also

SLL_INIT

4.7.2.4 sll_insert_first()

```
void sll_insert_first (
              struct sllist ** l,
              void * a )
```

Insert an element on the head of the list.

Parameters

`in,out`	*l*	An address to a pointer to the list.
`in`	*a*	The element.

4.7.2.5 sll_insert_last()

```
void sll_insert_last (
             struct sllist ** l,
             void * a )
```

Insert an element on the tail of the list.

Parameters

`in,out`	*l*	An address to a pointer to the list.
`in`	*a*	The element.

4.7.2.6 sll_remove_elm()

```
void* sll_remove_elm (
             struct sllist ** l,
             void * elm )
```

Remove the specified element of the list.

Parameters

`in,out`	*l*	An address to a pointer to the list.
`in`	*elm*	The element.

Returns

The element in case of success. NULL if the list is empty or the element doesn't exit.

4.7.2.7 sll_remove_first()

```
void* sll_remove_first (
             struct sllist ** l )
```

Remove the first element of the list.

The list node will be freed.

Parameters

in,out	*l*	An address to a pointer to the list.

Returns

The element in case of success. NULL if the list is empty.

4.7.2.8 sll_remove_last()

```
void* sll_remove_last (
            struct sllist ** l )
```

Remove the last element of the list.

The list node will be freed.

Parameters

in,out	*l*	An address to a pointer to the list.

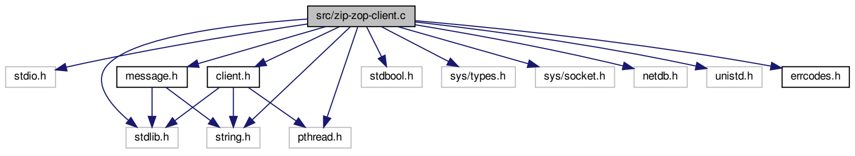
Returns

The element in case of success. NULL if the list is empty.

4.8 src/zip-zop-client.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <unistd.h>
#include <pthread.h>
#include "errcodes.h"
#include "message.h"
#include "client.h"
```

Include dependency graph for zip-zop-client.c:



Macros

- #define PORT "1234"

  The port where this application will be running.
- #define MESSAGE_LEN 2000

  Maximum length of a client message.

Functions

- bool check_args (int argc)

  Checks if the user enter the arguments in the correct manner.
- void print_usage (const char ∗name)

  Prints the correct usage of the program.
- void show_message (struct message ∗m)

  Displays a message in the screen.
- void ∗ listen_to_server_thread (void ∗client)

  Keeps listening to server messages.
- void ∗ speak_thread (void ∗client)

  Keeps reading messages from `stdin` *and send them to server.*
- struct addrinfo ∗ get_server_addr (const char ∗server_name)

  Gets the internet address of the server.
- int create_and_connect (struct addrinfo ∗addr)

  Attempts to create a socket to an internet address and connect to it in to a port.
- void server_introduction (struct client ∗c)

  Presents the client to the server.
- void communicate (const char ∗user_name, int sockfd)

  Manages the connection with a user and a server.
- int configure_as_client (const char ∗server_name)

  This function is responsible to make the initial configuration, so that this program can run as a client.
- int main (int argc, char ∗∗argv)

  The zip-zop-client.

4.8.1 Macro Definition Documentation

4.8.1.1 MESSAGE_LEN

```
#define MESSAGE_LEN 2000
```

Maximum length of a client message.

4.8.1.2 PORT

```
#define PORT "1234"
```

The port where this application will be running.

4.8.2 Function Documentation

4.8.2.1 check_args()

```
bool check_args (
             int argc )
```

Checks if the user enter the arguments in the correct manner.

Parameters

in	*argc*	Number of arguments.

Returns

`true` if the arguments are correct, `false` otherwise.

4.8.2.2 communicate()

```
void communicate (
             const char * user_name,
             int sockfd )
```

Manages the connection with a user and a server.

Given a username and a socket connected with the server, manages the connection, creating a thread to listen to incomming messages from the server, and another to read messages from the user and send them to the server.

Parameters

in	*user_name*	The username.
in	*sockfd*	The socket connected to the server.

4.8.2.3 configure_as_client()

```
int configure_as_client (
             const char * server_name )
```

This function is responsible to make the initial configuration, so that this program can run as a client.

Returns

A socket connected with the zip-zop-server. The user should be able to call `send()` and `recv()` in this socket.

4.8.2.4 create_and_connect()

```
int create_and_connect (
            struct addrinfo * addr )
```

Attempts to create a socket to an internet address and connect to it in to a port.

Parameters

in	*addr*	The internet address.

Returns

The socket in case os success. `-1` otherwise.

4.8.2.5 get_server_addr()

```
struct addrinfo* get_server_addr (
            const char * server_name )
```

Gets the internet address of the server.

Given the server name, this function will try to find an internet address to this server.

Parameters

in	*server_name*	The server name.

Returns

A pointer to a list of possibly valid server internet addresses.

4.8.2.6 listen_to_server_thread()

```
void* listen_to_server_thread (
            void * client )
```

Keeps listening to server messages.

This function will be executed by a thread that is responsable for keep checking if there is a new message from the server.

If there is an new message, the thread will display the message.

Parameters

`in`	*client*	A pointer to the client.

See also

show_message

4.8.2.7 main()

```
int main (
            int argc,
            char ** argv )
```

The zip-zop-client.

A TCP client that will connect with an instance of the zip-zop-server.

Parameters

`in`	*argc*	Number of arguments given by the user.
`in`	*argc*	An array of strings representing the arguments given by the user

Note

Usage: ./zip-zop-client <server_addr> <username>

4.8.2.8 print_usage()

```
void print_usage (
            const char * name )
```

Prints the correct usage of the program.

Parameters

`in`	*name*	The name of this program.

4.8.2.9 server_introduction()

```
void server_introduction (
            struct client * c )
```

Presents the client to the server.

This function sends everything that is needed to introduce the client to the server.

In this case only the client name is sent to the server.

Parameters

in	*c*	The client.

4.8.2.10 show_message()

```
void show_message (
             struct message * m )
```

Displays a message in the screen.

Parameters

in	*m*	The message.

4.8.2.11 speak_thread()

```
void* speak_thread (
             void * client )
```

Keeps reading messages from `stdin` and send them to server.

The message will be sent as a packet version of a struct message.

Parameters

in	*c*	The client that sent the message.

See also

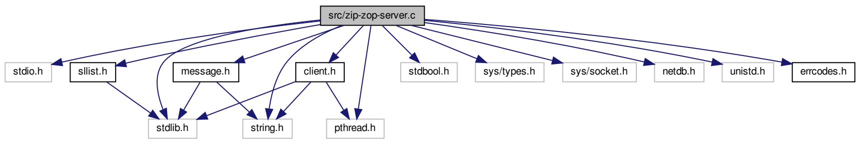
message_pack

4.9 src/zip-zop-server.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <stdbool.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <unistd.h>
#include <pthread.h>
#include "errcodes.h"
#include "message.h"
#include "client.h"
#include "sllist.h"
```

Include dependency graph for zip-zop-server.c:



Macros

- #define PORT "1234"

  The port where this application will be running.
- #define BACKLOG 10

  The number of clients that will be kept in the queue if the server is not ready for accepting them.
- #define CLIENT_NAME_LEN 100

  Maximum length of a client name.
- #define MESSAGE_LEN 2000

  Maximum length of a client message.

Functions

- void insert_client_concurrent (struct client ∗c)

  Carry out mutual exclusion and insert the new client on the list.
- struct client ∗ remove_client_concurrent (struct client ∗c)

  Carry out mutual exclusion and remove the new client on the list.
- void broadcast_client_message (struct client ∗c, const char ∗msg)

  Sends a message from one client to all clients.
- void broadcast_server_message (const char ∗msg)

  Sends a message from the server to all clients.
- void kill_client (struct client ∗c)

  Kill a client.
- void kill_all_clients (void)

  Kill all connected clients.
- void ∗ listen_to_client_thread (void ∗client)

  Keeps listening to client messages.
- void ∗ listen_to_commands_thread (void ∗arg)

  Keeps listening commands from stdin.
- void create_new_client (int sockfd)

  Create a new client and add it in the `CLIENT_LIST`.

- void ∗ accept_clients_thread (void ∗sock)

  Keeps on accepting new clients connections.
- struct addrinfo ∗ get_internet_addr (void)

  Find a set of possible internet addresses of localhost.
- int create_and_bind (struct addrinfo ∗addr)

  Attempts to create a socket and bind to a port with the given internet address.
- int configure_as_server (void)

  This function is responsible to make the initial configuration, so that this program can run as a server.
- int main (void)

  The zip-zop-server.

Variables

- struct sllist ∗ CLIENT_LIST = SLL_INIT()

  A singly linked list that will keep all the connected clients.
- pthread_mutex_t CLIENT_LIST_MUTEX

  The `CLIENT_LIST` mutex.

4.9.1 Macro Definition Documentation

4.9.1.1 BACKLOG

```
#define BACKLOG 10
```

The number of clients that will be kept in the queue if the server is not ready for accepting them.

4.9.1.2 CLIENT_NAME_LEN

```
#define CLIENT_NAME_LEN 100
```

Maximum length of a client name.

4.9.1.3 MESSAGE_LEN

```
#define MESSAGE_LEN 2000
```

Maximum length of a client message.

4.9.1.4 PORT

```
#define PORT "1234"
```

The port where this application will be running.

4.9.2 Function Documentation

4.9.2.1 accept_clients_thread()

```
void* accept_clients_thread (
             void * sock )
```

Keeps on accepting new clients connections.

Keeps listening for incoming connections, wen a new one arrives accepts it and instantiates a new client.

Parameters

in	*sock*	Adress to the socket used to listen to new connections.

4.9.2.2 broadcast_client_message()

```
void broadcast_client_message (
             struct client * c,
             const char * msg )
```

Sends a message from one client to all clients.

The message will be sent as a packet version of a struct message.

Parameters

in	*c*	The client that sent the message.
in	*msg*	The message content.

See also

message_pack

4.9.2.3 broadcast_server_message()

```
void broadcast_server_message (
             const char * msg )
```

Sends a message from the server to all clients.

The message will be sent as a packet version of a struct message.

Parameters

in	*msg*	The message content.

See also

message_pack

4.9.2.4 configure_as_server()

```
int configure_as_server (
             void  )
```

This function is responsible to make the initial configuration, so that this program can run as a server.

Returns

A socket in passive mode, that has the localhost address asigned to it. The user should be able to call accept() in this socket.

4.9.2.5 create_and_bind()

```
int create_and_bind (
             struct addrinfo * addr )
```

Attempts to create a socket and bind to a port with the given internet address.

Parameters

in	*addr*	The internet address.

Returns

The socket in case os success. `-1` otherwise.

4.9.2.6 create_new_client()

```
void create_new_client (
             int sockfd )
```

Create a new client and add it in the `CLIENT_LIST`.

Also broadcast everyone that the new client has entered the room.

Parameters

`in`	*sockfd*	The socket created in accept_clients_thread(), and that is used to communicate with the client that will be created.

See also

accept_clients_thread
CLIENT_LIST

4.9.2.7 get_internet_addr()

```
struct addrinfo* get_internet_addr (
             void  )
```

Find a set of possible internet addresses of localhost.

Returns

A list of addrinfo, wich contain the adresses.

4.9.2.8 insert_client_concurrent()

```
void insert_client_concurrent (
             struct client * c )
```

Carry out mutual exclusion and insert the new client on the list.

This function locks the `CLIENT_LIST_MUTEX` and inserts the client on the list, then unlocks the mutex.

Parameters

`in`	*c*	The client.

See also

CLIENT_LIST
CLIENT_LIST_MUTEX

4.9.2.9 kill_all_clients()

```
void kill_all_clients (
             void  )
```

Kill all connected clients.

Removes all clients from the `CLIENT_LIST`, destroy them and closes the connection.

See also

CLIENT_LIST

4.9.2.10 kill_client()

```
void kill_client (
             struct client * c )
```

Kill a client.

Removes a client from the `CLIENT_LIST`, destroys it and closes the connection.

Parameters

in	*c*	The client.

See also

CLIENT_LIST

4.9.2.11 listen_to_client_thread()

```
void* listen_to_client_thread (
             void * client )
```

Keeps listening to client messages.

This function will be executed by a thread that is responsable for keep checking if there is a new message from the client.

If there is an new message, the thread will execute the broadcast_client_message().

Parameters

in	*client*	A pointer to the client.

See also

broadcast_client_message

4.9.2.12 listen_to_commands_thread()

```
void* listen_to_commands_thread (
             void * arg )
```

Keeps listening commands from stdin.

This function will be executed by a thread responsible for listen to user commands.

Parameters

arg	An adress to the accept_clients_thread() thread, so it can cancel the thread when the server administrator executes the `/shutdown` command.

See also

accept_clients_thread

4.9.2.13 main()

```
int main (
             void  )
```

The zip-zop-server.

A TCP server that will accept connections from zip-zop-clients, hear its messages and broadcast them to all connected clients. Working as a chatroom.

4.9.2.14 remove_client_concurrent()

```
struct client* remove_client_concurrent (
             struct client * c )
```

Carry out mutual exclusion and remove the new client on the list.

This function locks the `CLIENT_LIST_MUTEX` and inserts the client on the list, then unlocks the mutex.

Parameters

in	*c*	The client.

Returns

The client just removed. NULL otherwise.

See also

CLIENT_LIST
CLIENT_LIST_MUTEX

4.9.3 Variable Documentation

4.9.3.1 CLIENT_LIST

```
struct sllist* CLIENT_LIST = SLL_INIT()
```

A singly linked list that will keep all the connected clients.

Warning

Mutual exclusion must be ensured before accessing this list.

See also

CLIENT_LIST_MUTEX

4.9.3.2 CLIENT_LIST_MUTEX

```
pthread_mutex_t CLIENT_LIST_MUTEX
```

The `CLIENT_LIST` mutex.

This is used to ensure mutual exclusion wen accessing the `CLIENT_LIST`, given the nature of the application where multiple threads might use the list.

See also

CLIENT_LIST

Index