

ECE297 Storage Server

0.2

Generated by Doxygen 1.8.1.2

Wed Jan 29 2014 13:44:28

Contents

1	Class Index	1
1.1	Class List	1
2	File Index	3
2.1	File List	3
3	Class Documentation	5
3.1	config_params Struct Reference	5
3.1.1	Detailed Description	5
3.2	storage_record Struct Reference	5
3.2.1	Detailed Description	6
4	File Documentation	7
4.1	client.c File Reference	7
4.1.1	Detailed Description	7
4.1.2	Function Documentation	7
4.1.2.1	main	7
4.2	encrypt_passwd.c File Reference	8
4.2.1	Detailed Description	8
4.3	server.c File Reference	8
4.3.1	Detailed Description	9
4.3.2	Function Documentation	9
4.3.2.1	handle_command	9
4.3.2.2	main	9
4.4	storage.c File Reference	10
4.4.1	Detailed Description	10
4.4.2	Function Documentation	10
4.4.2.1	storage_auth	10
4.4.2.2	storage_connect	11

4.4.2.3	storage_disconnect	11
4.4.2.4	storage_get	11
4.4.2.5	storage_set	11
4.5	storage.h File Reference	11
4.5.1	Detailed Description	13
4.5.2	Function Documentation	13
4.5.2.1	storage_auth	13
4.5.2.2	storage_connect	14
4.5.2.3	storage_disconnect	14
4.5.2.4	storage_get	14
4.5.2.5	storage_query	15
4.5.2.6	storage_set	15
4.6	utils.c File Reference	16
4.6.1	Detailed Description	17
4.6.2	Function Documentation	17
4.6.2.1	generate_encrypted_password	17
4.6.2.2	logger	17
4.6.2.3	read_config	17
4.6.2.4	recvline	18
4.6.2.5	sendall	18
4.7	utils.h File Reference	18
4.7.1	Detailed Description	19
4.7.2	Macro Definition Documentation	19
4.7.2.1	DBG	19
4.7.2.2	LOG	19
4.7.3	Function Documentation	19
4.7.3.1	generate_encrypted_password	19
4.7.3.2	logger	20
4.7.3.3	read_config	20
4.7.3.4	recvline	20
4.7.3.5	sendall	21

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

config_params	A struct to store config parameters	5
storage_record	Encapsulate the value associated with a key in a table	5

Chapter 2

File Index

2.1 File List

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

client.c	This file implements a "very" simple sample client	7
encrypt_passwd.c	This program implements a password encryptor	8
server.c	This file implements the storage server	8
storage.c	This file contains the implementation of the storage server interface as specified in storage.h	10
storage.h	This file defines the interface between the storage client and server	11
utils.c	This file implements various utility functions that are can be used by the storage server and client library	16
utils.h	This file declares various utility functions that are can be used by the storage server and client library	18

Chapter 3

Class Documentation

3.1 config_params Struct Reference

A struct to store config parameters.

```
#include <utils.h>
```

Public Attributes

- char [server_host](#) [[MAX_HOST_LEN](#)]
The hostname of the server.
- int [server_port](#)
The listening port of the server.
- char [username](#) [[MAX_USERNAME_LEN](#)]
The storage server's username.
- char [password](#) [[MAX_ENC_PASSWORD_LEN](#)]
The storage server's encrypted password.

3.1.1 Detailed Description

A struct to store config parameters.

Definition at line 49 of file [utils.h](#).

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

- [utils.h](#)

3.2 storage_record Struct Reference

Encapsulate the value associated with a key in a table.

```
#include <storage.h>
```

Public Attributes

- char [value](#) [[MAX_VALUE_LEN](#)]
This is where the actual value is stored.
- uintptr_t [metadata](#) [8]
A place to put any extra data.

3.2.1 Detailed Description

Encapsulate the value associated with a key in a table.

The metadata will be used later.

Definition at line 54 of file [storage.h](#).

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

- [storage.h](#)

Chapter 4

File Documentation

4.1 client.c File Reference

This file implements a "very" simple sample client.

```
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include "storage.h"
#include "utils.h"
```

Functions

- `int main (int argc, char *argv[])`
Start a client to interact with the storage server.

Variables

- `FILE * ClientFile`
This is just a minimal stub implementation. You should modify it according to your design.

4.1.1 Detailed Description

This file implements a "very" simple sample client. The client connects to the server, running at SERVERHOST:SERVERPORT and performs a number of storage_* operations. If there are errors, the client exists.

Definition in file [client.c](#).

4.1.2 Function Documentation

4.1.2.1 `int main (int argc, char * argv[])`

Start a client to interact with the storage server.

If connect is successful, the client performs a `storage_set/get()` on TABLE and KEY and outputs the results on stdout. Finally, it exists after disconnecting from the server.

Definition at line 24 of file `client.c`.

References `ClientFile`, `logger()`, `MAX_CONFIG_LINE_LEN`, `MAX_ENC_PASSWORD_LEN`, `MAX_HOST_LEN`, `MAX_KEY_LEN`, `MAX_TABLE_LEN`, `MAX_USERNAME_LEN`, `storage_auth()`, `storage_connect()`, `storage_disconnect()`, `storage_get()`, `storage_set()`, and `storage_record::value`.

4.2 `encrypt_passwd.c` File Reference

This program implements a password encryptor.

```
#include <stdlib.h>
#include <stdio.h>
#include "utils.h"
```

Functions

- void `print_usage()`
Print the usage to stdout.
- int `main` (int argc, char *argv[])

4.2.1 Detailed Description

This program implements a password encryptor.

Definition in file [encrypt_passwd.c](#).

4.3 `server.c` File Reference

This file implements the storage server.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <string.h>
#include <assert.h>
#include <signal.h>
#include "utils.h"
```

Macros

- `#define MAX_LISTENQUEUELEN 20`
The maximum number of queued connections.

Functions

- int [handle_command](#) (int sock, char *cmd)
Process a command from the client.
- int [main](#) (int argc, char *argv[])
Start the storage server.

Variables

- FILE * **serverlogger**
- char **stringmake** [[MAX_CMD_LEN](#)]

4.3.1 Detailed Description

This file implements the storage server. The storage server should be named "server" and should take a single command line argument that refers to the configuration file.

The storage server should be able to communicate with the client library functions declared in [storage.h](#) and implemented in [storage.c](#).

Definition in file [server.c](#).

4.3.2 Function Documentation

4.3.2.1 int [handle_command](#) (int *sock*, char * *cmd*)

Process a command from the client.

Parameters

<i>sock</i>	The socket connected to the client.
<i>cmd</i>	The command received from the client.

Returns

Returns 0 on success, -1 otherwise.

Definition at line 36 of file [server.c](#).

References [logger\(\)](#), and [sendall\(\)](#).

Referenced by [main\(\)](#).

4.3.2.2 int [main](#) (int *argc*, char * *argv*[])

Start the storage server.

This is the main entry point for the storage server. It reads the configuration file, starts listening on a port, and processes commands from clients.

Definition at line 62 of file [server.c](#).

References [handle_command\(\)](#), [logger\(\)](#), [MAX_CMD_LEN](#), [MAX_CONFIG_LINE_LEN](#), [MAX_LISTENQUEUELEN](#), [read_config\(\)](#), [recvline\(\)](#), [config_params::server_host](#), and [config_params::server_port](#).

4.4 storage.c File Reference

This file contains the implementation of the storage server interface as specified in [storage.h](#).

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include "storage.h"
#include "utils.h"
```

Functions

- void * [storage_connect](#) (const char *hostname, const int port)
Establish a connection to the server.
- int [storage_auth](#) (const char *username, const char *passwd, void *conn)
This is just a minimal stub implementation. You should modify it according to your design.
- int [storage_get](#) (const char *table, const char *key, struct [storage_record](#) *record, void *conn)
This is just a minimal stub implementation. You should modify it according to your design.
- int [storage_set](#) (const char *table, const char *key, struct [storage_record](#) *record, void *conn)
This is just a minimal stub implementation. You should modify it according to your design.
- int [storage_disconnect](#) (void *conn)
This is just a minimal stub implementation. You should modify it according to your design.

Variables

- FILE * [ClientFile](#)
This is just a minimal stub implementation. You should modify it according to your design.

4.4.1 Detailed Description

This file contains the implementation of the storage server interface as specified in [storage.h](#).

Definition in file [storage.c](#).

4.4.2 Function Documentation

4.4.2.1 int storage_auth (const char * username, const char * passwd, void * conn)

This is just a minimal stub implementation. You should modify it according to your design.

Authenticate the client's connection to the server.

Definition at line 70 of file storage.c.

References [ClientFile](#), [generate_encrypted_password\(\)](#), [logger\(\)](#), [MAX_CMD_LEN](#), [recvline\(\)](#), and [sendall\(\)](#).

Referenced by [main\(\)](#).

4.4.2.2 void* storage_connect (const char * *hostname*, const int *port*)

Establish a connection to the server.

Parameters

<i>hostname</i>	The IP address or hostname of the server.
<i>port</i>	The TCP port of the server.

Returns

If successful, return a pointer to a data structure that represents a connection to the server. Otherwise return NULL.

On error, errno will be set to one of the following, as appropriate: ERR_INVALID_PARAM, ERR_CONNECTION_FAIL, or ERR_UNKNOWN.

Definition at line 27 of file storage.c.

References ClientFile, logger(), and MAX_PORT_LEN.

Referenced by main().

4.4.2.3 int storage_disconnect (void * *conn*)

This is just a minimal stub implementation. You should modify it according to your design.

Close the connection to the server.

Definition at line 170 of file storage.c.

Referenced by main().

4.4.2.4 int storage_get (const char * *table*, const char * *key*, struct storage_record * *record*, void * *conn*)

This is just a minimal stub implementation. You should modify it according to your design.

Retrieve the value associated with a key in a table.

Definition at line 104 of file storage.c.

References ClientFile, logger(), MAX_CMD_LEN, recvline(), sendall(), and storage_record::value.

Referenced by main().

4.4.2.5 int storage_set (const char * *table*, const char * *key*, struct storage_record * *record*, void * *conn*)

This is just a minimal stub implementation. You should modify it according to your design.

Store a key/value pair in a table.

Definition at line 138 of file storage.c.

References ClientFile, logger(), MAX_CMD_LEN, recvline(), sendall(), and storage_record::value.

Referenced by main().

4.5 storage.h File Reference

This file defines the interface between the storage client and server.

```
#include <stdint.h>
```

Classes

- struct [storage_record](#)

Encapsulate the value associated with a key in a table.

Macros

- #define [MAX_CONFIG_LINE_LEN](#) 1024
Max characters in each config file line.
- #define [MAX_USERNAME_LEN](#) 64
Max characters of server username.
- #define [MAX_ENC_PASSWORD_LEN](#) 64
Max characters of server's encrypted password.
- #define [MAX_HOST_LEN](#) 64
Max characters of server hostname.
- #define [MAX_PORT_LEN](#) 8
Max characters of server port.
- #define [MAX_PATH_LEN](#) 256
Max characters of data directory path.
- #define [MAX_TABLES](#) 100
Max tables supported by the server.
- #define [MAX_RECORDS_PER_TABLE](#) 1000
Max records per table.
- #define [MAX_TABLE_LEN](#) 20
Max characters of a table name.
- #define [MAX_KEY_LEN](#) 20
Max characters of a key name.
- #define [MAX_CONNECTIONS](#) 10
Max simultaneous client connections.
- #define [MAX_COLUMNS_PER_TABLE](#) 10
Max columns per table.
- #define [MAX_COLNAME_LEN](#) 20
Max characters of a column name.
- #define [MAX_STRTYPE_SIZE](#) 40
Max SIZE of string types.
- #define [MAX_VALUE_LEN](#) 800
Max characters of a value.
- #define [ERR_INVALID_PARAM](#) 1
A parameter is not valid.
- #define [ERR_CONNECTION_FAIL](#) 2
Error connecting to server.
- #define [ERR_NOT_AUTHENTICATED](#) 3
Client not authenticated.
- #define [ERR_AUTHENTICATION_FAILED](#) 4

Client authentication failed.

- #define [ERR_TABLE_NOT_FOUND](#) 5

The table does not exist.

- #define [ERR_KEY_NOT_FOUND](#) 6

The key does not exist.

- #define [ERR_UNKNOWN](#) 7

Any other error.

- #define [ERR_TRANSACTION_ABORT](#) 8

Transaction abort error.

Functions

- void * [storage_connect](#) (const char *hostname, const int port)
Establish a connection to the server.
- int [storage_auth](#) (const char *username, const char *passwd, void *conn)
Authenticate the client's connection to the server.
- int [storage_get](#) (const char *table, const char *key, struct [storage_record](#) *record, void *conn)
Retrieve the value associated with a key in a table.
- int [storage_set](#) (const char *table, const char *key, struct [storage_record](#) *record, void *conn)
Store a key/value pair in a table.
- int [storage_query](#) (const char *table, const char *predicates, char **keys, const int max_keys, void *conn)
Query the table for records, and retrieve the matching keys.
- int [storage_disconnect](#) (void *conn)
Close the connection to the server.

4.5.1 Detailed Description

This file defines the interface between the storage client and server. The functions here should be implemented in [storage.c](#).

You should not modify this file, or else the code used to mark your implementation will break.

Definition in file [storage.h](#).

4.5.2 Function Documentation

4.5.2.1 int [storage_auth](#) (const char * *username*, const char * *passwd*, void * *conn*)

Authenticate the client's connection to the server.

Parameters

<i>username</i>	Username to access the storage server.
<i>passwd</i>	Password in its plain text form.
<i>conn</i>	A connection to the server.

Returns

Return 0 if successful, and -1 otherwise.

On error, `errno` will be set to `ERR_AUTHENTICATION_FAILED`.

Definition at line 70 of file `storage.c`.

References `ClientFile`, `generate_encrypted_password()`, `logger()`, `MAX_CMD_LEN`, `recvline()`, and `sendall()`.

Referenced by `main()`.

4.5.2.2 void* storage_connect (const char * *hostname*, const int *port*)

Establish a connection to the server.

Parameters

<i>hostname</i>	The IP address or hostname of the server.
<i>port</i>	The TCP port of the server.

Returns

If successful, return a pointer to a data structure that represents a connection to the server. Otherwise return `NULL`.

On error, `errno` will be set to one of the following, as appropriate: `ERR_INVALID_PARAM`, `ERR_CONNECTION_FAIL`, or `ERR_UNKNOWN`.

Definition at line 27 of file `storage.c`.

References `ClientFile`, `logger()`, and `MAX_PORT_LEN`.

Referenced by `main()`.

4.5.2.3 int storage_disconnect (void * *conn*)

Close the connection to the server.

Parameters

<i>conn</i>	A pointer to the connection structure returned in an earlier call to storage_connect() .
-------------	--

Returns

Return 0 if successful, and -1 otherwise.

On error, `errno` will be set to one of the following, as appropriate: `ERR_INVALID_PARAM`, `ERR_CONNECTION_FAIL`, or `ERR_UNKNOWN`.

Definition at line 170 of file `storage.c`.

Referenced by `main()`.

4.5.2.4 int storage_get (const char * *table*, const char * *key*, struct storage_record * *record*, void * *conn*)

Retrieve the value associated with a key in a table.

Parameters

<i>table</i>	A table in the database.
<i>key</i>	A key in the table.
<i>record</i>	A pointer to a record structure.
<i>conn</i>	A connection to the server.

Returns

Return 0 if successful, and -1 otherwise.

On error, `errno` will be set to one of the following, as appropriate: `ERR_INVALID_PARAM`, `ERR_CONNECTION_FAIL`, `ERR_TABLE_NOT_FOUND`, `ERR_KEY_NOT_FOUND`, `ERR_NOT_AUTHENTICATED`, or `ERR_UNKNOWN`.

The record with the specified key in the specified table is retrieved from the server using the specified connection. If the key is found, the record structure is populated with the details of the corresponding record. Otherwise, the record structure is not modified.

Definition at line 104 of file `storage.c`.

References `ClientFile`, `logger()`, `MAX_CMD_LEN`, `recvline()`, `sendall()`, and `storage_record::value`.

Referenced by `main()`.

4.5.2.5 int storage_query (const char * *table*, const char * *predicates*, char ** *keys*, const int *max_keys*, void * *conn*)

Query the table for records, and retrieve the matching keys.

Parameters

<i>table</i>	A table in the database.
<i>predicates</i>	A comma separated list of predicates.
<i>keys</i>	An array of strings where the keys whose records match the specified predicates will be copied. The array must have room for at least <code>max_keys</code> elements. The caller must allocate memory for this array.
<i>max_keys</i>	The size of the keys array.
<i>conn</i>	A connection to the server.

Returns

Return the number of matching keys (which may be more than `max_keys`) if successful, and -1 otherwise.

On error, `errno` will be set to one of the following, as appropriate: `ERR_INVALID_PARAM`, `ERR_CONNECTION_FAIL`, `ERR_TABLE_NOT_FOUND`, `ERR_KEY_NOT_FOUND`, `ERR_NOT_AUTHENTICATED`, or `ERR_UNKNOWN`.

Each predicate consists of a column name, an operator, and a value, each separated by optional whitespace. The operator may be a "=" for string types, or one of "<", ">", "=" for int and float types. An example of query predicates is "name = bob, mark > 90".

4.5.2.6 int storage_set (const char * *table*, const char * *key*, struct storage_record * *record*, void * *conn*)

Store a key/value pair in a table.

Parameters

<i>table</i>	A table in the database.
<i>key</i>	A key in the table.
<i>record</i>	A pointer to a record struture.
<i>conn</i>	A connection to the server.

Returns

Return 0 if successful, and -1 otherwise.

On error, errno will be set to one of the following, as appropriate: ERR_INVALID_PARAM, ERR_CONNECTION_FAIL, ERR_TABLE_NOT_FOUND, ERR_KEY_NOT_FOUND, ERR_NOT_AUTHENTICATED, or ERR_UNKNOWN.

The key and record are stored in the table of the database using the connection. If the key already exists in the table, the corresponding record is updated with the one specified here. If the key exists in the table and the record is NULL, the key/value pair are deleted from the table.

Definition at line 138 of file storage.c.

References ClientFile, logger(), MAX_CMD_LEN, recvline(), sendall(), and storage_record::value.

Referenced by main().

4.6 utils.c File Reference

This file implements various utility functions that are can be used by the storage server and client library.

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include "utils.h"
```

Functions

- int [sendall](#) (const int sock, const char *buf, const size_t len)
Keep sending the contents of the buffer until complete.
- int [recvline](#) (const int sock, char *buf, const size_t buflen)
Receive an entire line from a socket.
- int [process_config_line](#) (char *line, struct [config_params](#) *params)
Parse and process a line in the config file.
- int [read_config](#) (const char *config_file, struct [config_params](#) *params)
Read and load configuration parameters.
- void [logger](#) (FILE *file, char *message)
Generates a log message.
- char * [generate_encrypted_password](#) (const char *passwd, const char *salt)
Generates an encrypted password string using salt CRYPT_SALT.

4.6.1 Detailed Description

This file implements various utility functions that are can be used by the storage server and client library.

Definition in file [utils.c](#).

4.6.2 Function Documentation

4.6.2.1 `char* generate_encrypted_password (const char * passwd, const char * salt)`

Generates an encrypted password string using salt CRYPT_SALT.

Parameters

<i>passwd</i>	Password before encryption.
<i>salt</i>	Salt used to encrypt the password. If NULL default value DEFAULT_CRYPT_SALT is used.

Returns

Returns encrypted password.

Definition at line 135 of file `utils.c`.

References `DEFAULT_CRYPT_SALT`.

Referenced by `storage_auth()`.

4.6.2.2 `void logger (FILE * file, char * message)`

Generates a log message.

Parameters

<i>file</i>	The output stream
<i>message</i>	Message to log.

Definition at line 129 of file `utils.c`.

Referenced by `handle_command()`, `main()`, `storage_auth()`, `storage_connect()`, `storage_get()`, and `storage_set()`.

4.6.2.3 `int read_config (const char * config_file, struct config_params * params)`

Read and load configuration parameters.

Parameters

<i>config_file</i>	The name of the configuration file.
<i>params</i>	The structure where config parameters are loaded.

Returns

Return 0 on success, -1 otherwise.

Definition at line 104 of file `utils.c`.

References MAX_CONFIG_LINE_LEN, and process_config_line().

Referenced by main().

4.6.2.4 int recvline (const int sock, char * buf, const size_t buflen)

Receive an entire line from a socket.

In order to avoid reading more than a line from the stream, this function only reads one byte at a time. This is very inefficient, and you are free to optimize it or implement your own function.

Definition at line 37 of file utils.c.

Referenced by main(), storage_auth(), storage_get(), and storage_set().

4.6.2.5 int sendall (const int sock, const char * buf, const size_t len)

Keep sending the contents of the buffer until complete.

Returns

Return 0 on success, -1 otherwise.

The parameters mimic the send() function.

Definition at line 17 of file utils.c.

Referenced by handle_command(), storage_auth(), storage_get(), and storage_set().

4.7 utils.h File Reference

This file declares various utility functions that are can be used by the storage server and client library.

```
#include <stdio.h>
#include "storage.h"
#include <time.h>
```

Classes

- struct [config_params](#)
A struct to store config parameters.

Macros

- #define **LOGGING** 2
- #define [MAX_CMD_LEN](#) (1024 * 8)
The max length in bytes of a command from the client to the server.
- #define [LOG](#)(x) {printf x; fflush(stdout);}
A macro to log some information.
- #define [DBG](#)(x) {printf x; fflush(stdout);}
A macro to output debug information.

- `#define DEFAULT_CRYPT_SALT "xx"`
Default two character salt used for password encryption.

Functions

- `int sendall` (const int sock, const char *buf, const size_t len)
Keep sending the contents of the buffer until complete.
- `int recvline` (const int sock, char *buf, const size_t buflen)
Receive an entire line from a socket.
- `int read_config` (const char *config_file, struct config_params *params)
Read and load configuration parameters.
- `void logger` (FILE *file, char *message)
Generates a log message.
- `char * generate_encrypted_password` (const char *passwd, const char *salt)
Generates an encrypted password string using salt CRYPT_SALT.

4.7.1 Detailed Description

This file declares various utility functions that are can be used by the storage server and client library.

Definition in file [utils.h](#).

4.7.2 Macro Definition Documentation

4.7.2.1 `#define DBG(x) {printf x; fflush(stdout);}`

A macro to output debug information.

It is only enabled in debug builds.

Definition at line 43 of file [utils.h](#).

4.7.2.2 `#define LOG(x) {printf x; fflush(stdout);}`

A macro to log some information.

Use it like this: `LOG(("Hello %s", "world\n"))`

Don't forget the double parentheses, or you'll get weird errors!

Definition at line 33 of file [utils.h](#).

4.7.3 Function Documentation

4.7.3.1 `char* generate_encrypted_password (const char * passwd, const char * salt)`

Generates an encrypted password string using salt CRYPT_SALT.

Parameters

<i>passwd</i>	Password before encryption.
<i>salt</i>	Salt used to encrypt the password. If NULL default value DEFAULT_CRYPT_SALT is used.

Returns

Returns encrypted password.

Definition at line 135 of file `utils.c`.

References `DEFAULT_CRYPT_SALT`.

Referenced by `storage_auth()`.

4.7.3.2 void logger (FILE * *file*, char * *message*)

Generates a log message.

Parameters

<i>file</i>	The output stream
<i>message</i>	Message to log.

Definition at line 129 of file `utils.c`.

Referenced by `handle_command()`, `main()`, `storage_auth()`, `storage_connect()`, `storage_get()`, and `storage_set()`.

4.7.3.3 int read_config (const char * *config_file*, struct config_params * *params*)

Read and load configuration parameters.

Parameters

<i>config_file</i>	The name of the configuration file.
<i>params</i>	The structure where config parameters are loaded.

Returns

Return 0 on success, -1 otherwise.

Definition at line 104 of file `utils.c`.

References `MAX_CONFIG_LINE_LEN`, and `process_config_line()`.

Referenced by `main()`.

4.7.3.4 int recvline (const int *sock*, char * *buf*, const size_t *buflen*)

Receive an entire line from a socket.

Returns

Return 0 on success, -1 otherwise.

In order to avoid reading more than a line from the stream, this function only reads one byte at a time. This is very inefficient, and you are free to optimize it or implement your own function.

Definition at line 37 of file `utils.c`.

Referenced by `main()`, `storage_auth()`, `storage_get()`, and `storage_set()`.

4.7.3.5 `int sendall (const int sock, const char * buf, const size_t len)`

Keep sending the contents of the buffer until complete.

Returns

Return 0 on success, -1 otherwise.

The parameters mimic the `send()` function.

Definition at line 17 of file `utils.c`.

Referenced by `handle_command()`, `storage_auth()`, `storage_get()`, and `storage_set()`.

Index

- client.c, [7](#)
 - main, [7](#)
- config_params, [5](#)
- DBG
 - utils.h, [19](#)
- encrypt_passwd.c, [8](#)
- generate_encrypted_password
 - utils.c, [17](#)
 - utils.h, [19](#)
- handle_command
 - server.c, [9](#)
- LOG
 - utils.h, [19](#)
- logger
 - utils.c, [17](#)
 - utils.h, [20](#)
- main
 - client.c, [7](#)
 - server.c, [9](#)
- read_config
 - utils.c, [17](#)
 - utils.h, [20](#)
- recvline
 - utils.c, [18](#)
 - utils.h, [20](#)
- sendall
 - utils.c, [18](#)
 - utils.h, [20](#)
- server.c, [8](#)
 - handle_command, [9](#)
 - main, [9](#)
- storage.c, [10](#)
 - storage_auth, [10](#)
 - storage_connect, [10](#)
 - storage_disconnect, [11](#)
 - storage_get, [11](#)
 - storage_set, [11](#)
- storage.h, [11](#)
 - storage_auth, [13](#)
 - storage_connect, [14](#)
 - storage_disconnect, [14](#)
 - storage_get, [14](#)
 - storage_query, [15](#)
 - storage_set, [15](#)
- storage_auth
 - storage.c, [10](#)
 - storage.h, [13](#)
- storage_connect
 - storage.c, [10](#)
 - storage.h, [14](#)
- storage_disconnect
 - storage.c, [11](#)
 - storage.h, [14](#)
- storage_get
 - storage.c, [11](#)
 - storage.h, [14](#)
- storage_query
 - storage.h, [15](#)
- storage_record, [5](#)
- storage_set
 - storage.c, [11](#)
 - storage.h, [15](#)
- utils.c, [16](#)
 - generate_encrypted_password, [17](#)
 - logger, [17](#)
 - read_config, [17](#)
 - recvline, [18](#)
 - sendall, [18](#)
- utils.h, [18](#)
 - DBG, [19](#)
 - generate_encrypted_password, [19](#)
 - LOG, [19](#)
 - logger, [20](#)
 - read_config, [20](#)
 - recvline, [20](#)
 - sendall, [20](#)