ECE297 Storage Server

0.2

Generated by Doxygen 1.7.1

Thu Mar 28 2013 12:40:54

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 config_params Struct Reference

A struct to store config parameters. Includes table names and the struct table_info.

```
#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.

- char **table** [MAX_TABLES][MAX_TABLE_LEN]
- struct table_info **tableInfo** [100]
- char **key** [MAX_RECORDS_PER_TABLE][MAX_KEY_LEN]
- char **value** [MAX_RECORDS_PER_TABLE][MAX_VALUE_LEN]
- int **index**

3.1.1 Detailed Description

A struct to store config parameters. Includes table names and the struct table_info.

Definition at line 62 of file utils.h.

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

- utils.h

3.2 head Struct Reference

A header for our database.

Public Attributes

- struct table ∗ **head**

3.2.1 Detailed Description

A header for our database.

Definition at line 74 of file server.c.

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

- server.c

3.3 key Struct Reference

Simply stores the next key in the list and the value (each key is a struct).

Public Attributes

- char **name** [MAX_KEY_LEN]
- struct key ∗ **next**
- struct key ∗ **back**
- char **value** [MAX_VALUE_LEN]
- char **value_per_col** [MAX_COLUMNS_PER_TABLE][100]

3.3.1 Detailed Description

Simply stores the next key in the list and the value (each key is a struct).

Definition at line 61 of file server.c.

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

- server.c

3.4 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.

3.4.1 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

3.5 table Struct Reference

A struct a linked list to store the table's configuration paramaters.

Public Attributes

- struct key ∗ **head**

- struct table ∗ **back**
- struct table ∗ **next**
- char **name** [MAX_TABLE_LEN]
- char **col_name** [MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN]
- char **col_type** [MAX_COLUMNS_PER_TABLE][10]
- int **number_records**

3.5.1 Detailed Description

A struct a linked list to store the table's configuration paramaters.

Definition at line 34 of file server.c.

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

- server.c

3.6 table_info Struct Reference

A struct to store each table's configuration type.

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

Public Attributes

- char **table_names** [MAX_TABLE_LEN]
- char **name_for_column** [MAX_COLUMNS_PER_TABLE][MAX_-COLNAME_LEN]
- char **type_for_name** [MAX_COLUMNS_PER_TABLE][10]

3.6.1 Detailed Description

A struct to store each table's configuration type.

Definition at line 48 of file utils.h.

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

- utils.h

3.7 temp_table_info Struct Reference

A struct to store EACH table's paramters.

Public Attributes

- char **col_name** [MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN]
- char **col_type** [MAX_COLUMNS_PER_TABLE][10]
- char **col_value** [10][800]
- char **operators** [10][10]

3.7.1 Detailed Description

A struct to store EACH table's paramters.

Definition at line 49 of file server.c.

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

- server.c

Chapter 4

File Documentation

4.1 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[ ])

4.1.1 Detailed Description

This program implements a password encryptor.

Definition in file encrypt_passwd.c.

4.2 loghelp.h File Reference

this file is used to set the logging constant throughout all of the files

Defines

- #define **LOGGING** 1

Variables

- FILE ∗ **clientlog**

4.2.1 Detailed Description

this file is used to set the logging constant throughout all of the files

Definition in file loghelp.h.

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"
#include "loghelp.h"
#include <errno.h>
```

Classes

- struct table

  A struct a linked list to store the table's configuration paramaters.

- struct temp_table_info

  A struct to store EACH table's paramters.

- struct key

  Simply stores the next key in the list and the value (each key is a struct).

- struct head

  A header for our database.

Defines

- #define **MAX_SIZE_LENGTH** 800
- #define MAX_LISTENQUEUELEN 20

  The maximum number of queued connections.

Functions

- struct table ∗ create_table_bare (char ∗name_table, struct table_info ∗temp)

  A function that dynamically creates a bare table given its name and table configurations.

- struct table ∗ find_table (char ∗name, struct head ∗head)

  a function to find the table stored inside of our database

- struct key ∗ find_key (char ∗name, struct table ∗table)

  function to find the key stored inside of our database

- struct table ∗ **create_table** (char ∗name_table, char ∗name_key, char ∗value)
- struct key ∗ create_key (char ∗name_key, char ∗value, struct temp_table_info ∗temp)

  A function that dynamically creates a key given its key name, value.

- void insert_table (struct table ∗new_table, struct head ∗head)

  a function that inserts a table (new_table) inside of our database. Also inserts it into our structs

- void insert_key (struct key ∗new_key, struct table ∗table)

  a function that inserts a key (new_key) inside of our database. Also inserts it into our structs

- void **delete_table** (struct table ∗target_table, struct table ∗table)
- void delete_key (struct key ∗target_key, struct table ∗table)

  function that deletes the key from the linked list and sets the next/back value of respective keys in the list

- int set_function (char ∗table_name, char ∗key_name,char ∗value, struct head ∗head, struct temp_table_info ∗temp)

  a function that sets the value of a key based on the value sent. Function is called from handle_command

- int get_function (char ∗table_name, char ∗key_name, char ∗value, struct head ∗head)

  a function to get (retreive a value) from our database. Function is called from handle_command

- int compare_formats (struct temp_table_info ∗temp_info, struct table ∗target_-table)

  PARSING///.

- int query_parser (char ∗line, struct temp_table_info ∗temp_info)

  a function that parses the query predicates. Function called from handle command

- int input_parser (char ∗line, struct temp_table_info ∗temp_info)

  parses the user's input for a set command and saves it's format in temp_info

- int compare_query_format (struct temp_table_info ∗temp_info, struct table ∗target_table)

  function that compares if the format of the input (for storage_query) matches the columns/types of the table's format

- struct key ∗ check_key (struct key ∗target_key, int index_col, char ∗operators, char ∗type, char ∗value)

  a function that checks if the key's value satisfies the condition outlined by the operator

- int **table_query** (struct table ∗target_table, struct temp_table_info ∗col_info, struct key ∗key_array[50])
- int handle_command (int sock, char ∗cmd, FILE ∗∗log, struct config_params ∗params, struct head ∗head)

  Handle command takes in a sentence from the client and performs accordingly.

- int main (int argc, char ∗argv[ ])

  Start the storage server.

- int table_query (struct table ∗target_table, struct temp_table_info ∗col_info, struct key ∗key_array[ ])

  takes a set of conditions (col_info) and finds all the keys in the supplied table that match all the conditions

Variables

- int **didAuthenticate**

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.

At the bottom of the server file, we have implemented all of the functions needed for our database, along with the query function

Definition in file server.c.

4.3.2 Function Documentation

4.3.2.1 struct key ∗ check_key (struct key ∗ *target_key,* int *index_col,* char ∗ *operators,* char ∗ *type,* char ∗ *value*) `[read]`

a function that checks if the key's value satisfies the condition outlined by the operator

Parameters

target_key	is the key checked
index_col	holds the correct index for the key's column/value to match the provided value's type
operators	condition to be checked

Returns

returns the key struct if valid, null if not

Definition at line 1217 of file server.c.

Referenced by table_query().

4.3.2.2 int compare_formats (struct temp_table_info ∗ *temp_info,* struct table ∗ *target_table*)

PARSING///.

function that compares the formats of the user's input with the expected format in the table (used for storage_set)

Parameters

temp_info contains the format of the columns/types/values of the user input

target_table contains the format of the table, this is compared with the temp provided in temp_info

Returns

returns -1 for failure, and 0 for success

Definition at line 1008 of file server.c.

Referenced by set_function().

4.3.2.3 int compare_query_format (struct temp_table_info ∗ *temp_info,* struct table ∗ *target_table*)

function that compares if the format of the input (for storage_query) matches the columns/types of the table's format

Parameters

temp_info a pointer to a struct that contains the format of the input sent

target_table contains the format of the table, this is compared with the temp provided in temp_info

Returns

returns -1 for failure, and 0 for success

Definition at line 964 of file server.c.

Referenced by handle_command().

4.3.2.4 struct key ∗ create_key (char ∗ *name_key,* char ∗ *value,* struct temp_table_info ∗ *temp*) `[read]`

A function that dynamically creates a key given its key name, value.

Parameters

name_key name of the key

value value to place inside the key

temp struct of the table info

Returns

struct to the key created

Definition at line 691 of file server.c.

Referenced by set_function().

4.3.2.5 struct table ∗ create_table_bare (char ∗ *name_table,* struct table_info ∗ *temp*) `[read]`

A function that dynamically creates a bare table given its name and table configurations.

Parameters

name_table name of the table

table_info information for each table inside the config file

Returns

struct to the table created

Definition at line 657 of file server.c.

Referenced by main().

4.3.2.6 void delete_key (struct key ∗ *target_key,* struct table ∗ *table*)

function that deletes the key from the linked list and sets the next/back value of respective keys in the list

Parameters

target_key is the key to be deleted

table is the table the key is contained in

Returns

nothing

Definition at line 818 of file server.c.

Referenced by set_function().

4.3.2.7 struct key ∗ find_key (char ∗ *name,* struct table ∗ *table*) `[read]`

function to find the key stored inside of our database

Parameters

name of the table

head structure of the head to the database

Returns

struct of the table if found, or null if not found

Definition at line 609 of file server.c.

Referenced by get_function(), and set_function().

4.3.2.8 struct table ∗ find_table (char ∗ *name,* struct head ∗ *head*) `[read]`

a function to find the table stored inside of our database

Parameters

name of the table

head structure of the head to the database

Returns

struct of the table if found, or null if not found

Definition at line 588 of file server.c.

Referenced by get_function(), handle_command(), and set_function().

4.3.2.9 int get_function (char ∗ *table_name,* char ∗ *key_name,* char ∗ *value,* struct head ∗ *head*)

a function to get (retreive a value) from our database. Function is called from handle_-command

Parameters

table_name name of the table we must get from

key_name name of the key to retreive from

value returns the value inside

head header to the database

Returns

0 for success, 1 for key not found, 2 table not found

Definition at line 927 of file server.c.

References find_key(), and find_table().

Referenced by handle_command().

4.3.2.10 int handle_command (int *sock,* char ∗ *cmd,* FILE ∗∗ *log,* struct config_params ∗ *params,* struct head ∗ *head*)

Handle command takes in a sentence from the client and performs accordingly.

Parameters

sock The socket connected to the client.

cmd The command received from the client.

log a pointer to the pointer of the declaration of the file to log

params a pointer to the struct holding the properties inside the config file

head a pointer to the head of our database

Returns

Returns 0 on success, -1 otherwise.

Definition at line 124 of file server.c.

References compare_query_format(), find_table(), get_function(), input_parser(), logger(), MAX_VALUE_LEN, config_params::password, query_parser(), sendall(), set_function(), table_query(), and config_params::username.

Referenced by main().

4.3.2.11 int input_parser (char ∗ *line,* struct temp_table_info ∗ *temp_info*)

parses the user's input for a set command and saves it's format in temp_info

Parameters

line to parse

temp_info contains the saved info and format

Returns

returns -1 for failure, and 0 for success

Definition at line 1155 of file server.c.

Referenced by handle_command().

4.3.2.12 void insert_key (struct key ∗ *new_key,* struct table ∗ *table*)

a function that inserts a key (new_key) inside of our database. Also inserts it into our structs

Parameters

new_key a struct to the new key

table a struct to the table

Returns

nothing

Definition at line 750 of file server.c.

Referenced by set_function().

4.3.2.13 void insert_table (struct table ∗ *new_table,* struct head ∗ *head*)

a function that inserts a table (new_table) inside of our database. Also inserts it into our structs

Parameters

new_table a struct pointing to the table

head a struct pointing to the head of our database

Returns

nothing

Definition at line 719 of file server.c.

Referenced by main().

4.3.2.14 int main (int *argc,* char ∗ *argv[ ]*)

Start the storage server.

- starts the connection and ports

- it opens the file to log, processes the time output
- creates the tables and keys for the database
- processes the config file
- calls handle command

Returns

returns 0 for success, -1 for failure

Definition at line 406 of file server.c.

References create_table_bare(), handle_command(), insert_table(), logger(), MAX_-CMD_LEN, MAX_LISTENQUEUELEN, read_config(), recvline(), config_-params::server_host, and config_params::server_port.

4.3.2.15 int query_parser (char ∗ *line,* struct temp_table_info ∗ *temp_info*)

a function that parses the query predicates. Function called from handle command

Parameters

line to parse

temp_info stores the parsed information and format

Returns

returns -1 for failure, and 0 for success

Definition at line 1053 of file server.c.

Referenced by handle_command().

4.3.2.16 int set_function (char ∗ *table_name,* char ∗ *key_name,* char ∗ *value,* struct head ∗ *head,* struct temp_table_info ∗ *temp*)

a function that sets the value of a key based on the value sent. Function is called from handle_command

Parameters

table_name name of the table we must set into

key_name name of the key to input

value value to be set

head header to the database (head of table chain)

temp a struct that contains the format of the input, used to check if the format matches the table's format

Returns

0 for success, 1 for key not found, 2 table not found, 3 unknown, 4 invalid format of input (types do no match columns)

Definition at line 860 of file server.c.

References compare_formats(), create_key(), delete_key(), find_key(), find_table(), insert_key(), and MAX_RECORDS_PER_TABLE.

Referenced by handle_command().

4.3.2.17 int table_query (struct table ∗ *target_table,* struct temp_table_info ∗ *col_info,* struct key ∗ *key_array[ ]*)

takes a set of conditions (col_info) and finds all the keys in the supplied table that match all the conditions

Parameters

target_table a struct to the configuartion of that particular table

col_info holds the parsed conditions (from query_parser)

key_array holds the keys that satisfy all the conditions

Returns

returns -1 for failure, and 0 for success

filled Arr_Keys with the keys that satisfied at least one of the conditions now checking which keys match all conditions

Definition at line 1250 of file server.c.

References check_key().

Referenced by handle_command().

4.4 storage.c File Reference

This file contains the implementation of the storage server interface as specified in storage.h.

`#include <stdlib.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"
#include "loghelp.h"
#include <errno.h>
```

Functions

- int tablecheck (char ∗string)

  receives a string and checks whether it's valid. Standard given according to the M2 document

- int keycheck (char ∗string)

  receives a string and checks whether it's valid. Standard given according to the M2 document

- int valuecheck (char ∗string)

  receives a string and checks whether it's valid. Standard given according to the M2 document

- void ∗ storage_connect (const char ∗hostname, const int port)

  performs a basic error check and after everything has passed, passes the information onto server via sendall after the execution of this function, we will be conencted with our server

- int storage_auth (const char ∗username, const char ∗passwd, void ∗conn)

  performs a basic error check and after everything has passed, passes the information onto server via sendall after the execution of this function, we will be authenticated with the server

- int storage_get (const char ∗table, const char ∗key, struct storage_record ∗record, void ∗conn)

  performs a basic error check and after everything has passed, passes the information onto server via sendall after the execution of this function, we will have the values found inside the table

- int storage_set (const char ∗table, const char ∗key, struct storage_record ∗record, void ∗conn)

  performs a basic error check and after everything has passed, passes the information onto server via sendall after the execution of this function, we will have set a key along with its values inside the database

- int storage_query (const char ∗table, const char ∗predicates, char ∗∗keys, const int max_keys, void ∗conn)

  performs a basic error check and after everything has passed, passes the information onto server via sendall after the execution of this function, we will have all of the keys that pass the operations inside predicates

- int storage_disconnect (void ∗conn)

  performs a basic error check and after everything has passed, disconnects from the server

Variables

- int **didConnect**
- int **didAuthenticate**

4.4.1 Detailed Description

This file contains the implementation of the storage server interface as specified in storage.h. performs all of the sending and receiving with the server, also performs a standard error check on the commands given from client

Definition in file storage.c.

4.4.2 Function Documentation

4.4.2.1 int keycheck (char ∗ *string*)

receives a string and checks whether it's valid. Standard given according to the M2 document

Parameters

string is what we want to make sure is a valid key type

Returns

return 0 for success, 1 for failure

Definition at line 49 of file storage.c.

Referenced by storage_get(), and storage_set().

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

performs a basic error check and after everything has passed, passes the information onto server via sendall after the execution of this function, we will be authenticated with the server

Authenticate the client's connection to the server.

Parameters

username the name of our server we want to join

passwd the password for the server network

conn a pointer to the connection status of the server

Returns

return 0 for success, -1 for failure

Definition at line 139 of file storage.c.

References generate_encrypted_password(), recvline(), and sendall().

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

performs a basic error check and after everything has passed, passes the information onto server via sendall after the execution of this function, we will be conenected with our server

Establish a connection to the server.

Parameters

hostname is the name of the host we want to connect with

port is the port number we want to connect with

return 0 for success, -1 for failure

Definition at line 88 of file storage.c.

References logger().

4.4.2.4 int storage_disconnect (void ∗ *conn*)

performs a basic error check and after everything has passed, disconnects from the server

Close the connection to the server.

Parameters

conn a pointer to the connection status of the server

Returns

return 0 for success, -1 for failure

Definition at line 368 of file storage.c.

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

performs a basic error check and after everything has passed, passes the information onto server via sendall after the execution of this function, we will have the values found inside the table

Retrieve the value associated with a key in a table.

Parameters

table name of the table we want to get from

key name of the key

record is the struct holding the configuration parameters of everything inside the configuration file

conn a pointer to the connection status of the server

Returns

return 0 for success, -1 for failure

Definition at line 176 of file storage.c.

References keycheck(), recvline(), sendall(), tablecheck(), and storage_record::value.

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

performs a basic error check and after everything has passed, passes the information onto server via sendall after the execution of this function, we will have all of the keys that pass the operations inside predicates

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

Parameters

table name of the table we want to query from

key a member address of the pointer for the string of keys

max_keys is the max number of keys storage query is supposed to return

conn a pointer to the connection status of the server

Returns

return 0 for success, -1 for failure

Definition at line 302 of file storage.c.

References recvline(), sendall(), and tablecheck().

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

performs a basic error check and after everything has passed, passes the information onto server via sendall after the execution of this function, we will have set a key along with its values inside the database

Store a key/value pair in a table.

Parameters

table name of the table we want to set to

key name of the key we want to set

record is the struct holding the configuration parameters of everything inside the configuration file

conn a pointer to the connection status of the server

Returns

return 0 for success, -1 for failure

Definition at line 234 of file storage.c.

References keycheck(), recvline(), sendall(), tablecheck(), and storage_record::value.

4.4.2.8 int tablecheck (char ∗ *string*)

receives a string and checks whether it's valid. Standard given according to the M2 document

Parameters

string,string to perform error check on

Returns

return 0 for success, 1 for failure

Definition at line 32 of file storage.c.

Referenced by storage_get(), storage_query(), and storage_set().

4.4.2.9 int valuecheck (char * *string*)

receives a string and checks whether it's valid. Standard given according to the M2 document

Parameters

string is what we want to make sure is a valid value type

Returns

return 0 for success, 1 for failure

Definition at line 66 of file storage.c.

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.

Defines

- #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

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

Returns

Return 0 if successful, and -1 otherwise.

On error, errno will be set to ERR_AUTHENTICATION_FAILED.

Authenticate the client's connection to the server.

Parameters

username	the name of our server we want to join
passwd	the password for the server network
conn	a pointer to the connection status of the server

Returns

return 0 for success, -1 for failure

Definition at line 139 of file storage.c.

References generate_encrypted_password(), recvline(), and sendall().

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

Establish a connection to the server.

Parameters

hostname The IP address or hostname of the server.

port 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.

Establish a connection to the server.

Parameters

hostname is the name of the host we want to connect with

port is the port number we want to connect with

return 0 for success, -1 for failure

Definition at line 88 of file storage.c.

References logger().

4.5.2.3 int storage_disconnect (void ∗ *conn*)

Close the connection to the server.

Parameters

conn 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.

Close the connection to the server.

Parameters

conn a pointer to the connection status of the server

Returns

return 0 for success, -1 for failure

Definition at line 368 of file storage.c.

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

table A table in the database.

key A key in the table.

record A pointer to a record struture.

conn 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.

Retrieve the value associated with a key in a table.

Parameters

table name of the table we want to get from

key name of the key

record is the struct holding the configuration parameters of everything inside the configuration file

conn a pointer to the connection status of the server

Returns

return 0 for success, -1 for failure

Definition at line 176 of file storage.c.

References keycheck(), recvline(), sendall(), tablecheck(), and storage_record::value.

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

table A table in the database.

predicates A comma separated list of predicates.

keys An array of strings where the keys whose records match the specified predicates will be copied. The array must have room for at least max_keys elements. The caller must allocate memory for this array.

max_keys The size of the keys array.

conn 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".

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

Parameters

table name of the table we want to query from

key a member address of the pointer for the string of keys

max_keys is the max number of keys storage query is supposed to return

conn a pointer to the connection status of the server

Returns

return 0 for success, -1 for failure

Definition at line 302 of file storage.c.

References recvline(), sendall(), and tablecheck().

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

table A table in the database.
key A key in the table.
record A pointer to a record struture.
conn 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.

Store a key/value pair in a table.

Parameters

table name of the table we want to set to
key name of the key we want to set
record is the struct holding the configuration parameters of everything inside the configuration file
conn a pointer to the connection status of the server

Returns

return 0 for success, -1 for failure

Definition at line 234 of file storage.c.

References keycheck(), recvline(), sendall(), tablecheck(), and storage_record::value.

4.6 utils.c File Reference

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

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include "utils.h"
#include "loghelp.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, int ∗host, int ∗port, int ∗user, int ∗pass)

  takes a line passed in from the read configuration function and stores it in the appropriate structs

- int read_config (const char ∗config_file, struct config_params ∗params)

  opens and reads the entire config file, parsing and storing them to structs used in other files

- void logger (FILE ∗file, char ∗message)

  either writes to stdout, a file or does nothing according to the LOGGING constant set in the loghelp.h file

- 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 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

passwd Password before encryption.

salt Salt used to encrypt the password. If NULL default value DEFAULT_-CRYPT_SALT is used.

Returns

Returns encrypted password.

Definition at line 291 of file utils.c.

References DEFAULT_CRYPT_SALT.

Referenced by storage_auth().

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

either writes to stdout, a file or does nothing according to the LOGGING constant set in the loghelp.h file

Generates a log message.

Parameters

file is a pointer to the file created to log to

message is what we want to write to either the file, stdout or

Returns

nothing

Definition at line 280 of file utils.c.

Referenced by handle_command(), main(), and storage_connect().

4.6.2.3 int process_config_line (char ∗ *line,* struct config_params ∗ *params,* int ∗ *host,* int ∗ *port,* int ∗ *user,* int ∗ *pass*)

takes a line passed in from the read configuration function and stores it in the appropriate structs

Parameters

line is each line inside the config file
params is the struct that will hold the entire config file... split into arrays
host is the counter used through each function call -used for error checking
port is the counter used through each function call -used for error checking
user is the counter used through each function call -used for error checking
pass is the counter used through each function call -used for error checking

Returns

returns 0 for success, -1 for failure

Definition at line 77 of file utils.c.

References MAX_TABLE_LEN, config_params::password, config_params::server_host, config_params::server_port, and config_params::username.

Referenced by read_config().

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

opens and reads the entire config file, parsing and storing them to structs used in other files

Read and load configuration parameters.

Parameters

config_file is the name of the config file being parsed
params is a structure that holds everything inside the configuration file

Returns

returns 0 for success, -1 for failure

Definition at line 234 of file utils.c.

References process_config_line().

Referenced by main().

4.6.2.5 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 39 of file utils.c.

Referenced by main(), storage_auth(), storage_get(), storage_query(), and storage_-set().

4.6.2.6 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 19 of file utils.c.

Referenced by handle_command(), storage_auth(), storage_get(), storage_query(), 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"
```

Classes

- struct table_info

  A struct to store each table's configuration type.

- struct config_params

  A struct to store config parameters. Includes table names and the struct table_info.

Defines

- #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 Define 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 41 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 31 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

- ***passwd*** Password before encryption.
- ***salt*** Salt used to encrypt the password. If NULL default value DEFAULT_-CRYPT_SALT is used.

Returns

Returns encrypted password.

Definition at line 291 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

file The output stream
message Message to log.

Generates a log message.

Parameters

file is a pointer to the file created to log to
message is what we want to write to either the file, stdout or

Returns

nothing

Definition at line 280 of file utils.c.

Referenced by handle_command(), main(), and storage_connect().

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

Read and load configuration parameters.

Parameters

config_file The name of the configuration file./homes/l/loboweyl/Documents/storage/src/server.c:117: undefined reference to ‘set_function’
params The structure where config parameters are loaded.

Returns

Return 0 on success, -1 otherwise.

Read and load configuration parameters.

Parameters

config_file is the name of the config file being parsed
params is a structure that holds everything inside the configuration file

Returns

returns 0 for success, -1 for failure

Definition at line 234 of file utils.c.

References 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 39 of file utils.c.

Referenced by main(), storage_auth(), storage_get(), storage_query(), 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 19 of file utils.c.

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

Index