ECE297 Storage Server

0.2

Generated by Doxygen 1.8.1.2

Sun Apr 6 2014 00:35:26

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 _ThreadInfo Struct Reference

Structure storing information for each thread.

Public Attributes

- struct sockaddr_in **clientaddr**
- socklen_t **clientaddrlen**
- int **clientsock**
- pthread_t **theThread**

3.1.1 Detailed Description

Structure storing information for each thread.

Definition at line 33 of file server.c.

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

- server.c

3.2 column_struct Struct Reference

Public Attributes

- char **column_name** [MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN]
- char **value** [MAX_COLUMNS_PER_TABLE][MAX_VALUE_LEN]
- int **type** [MAX_COLUMNS_PER_TABLE]

3.2.1 Detailed Description

Definition at line 65 of file utils.h.

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

- utils.h

3.3 config_params Struct Reference

struct for storing information about the current configuration file

`#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.
- int concurrency

  Concurrency setting number.
- char username [MAX_USERNAME_LEN]

  The storage server's username.
- char pass_ [MAX_ENC_PASSWORD_LEN]

  The user's password.
- char **table_name** [MAX_TABLES][MAX_TABLE_LEN]
- struct table_params **tablepara** [MAX_TABLES]
- bool **duplicateparam**
- bool **duplicatetable**

3.3.1 Detailed Description

struct for storing information about the current configuration file

Definition at line 127 of file utils.h.

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

- utils.h

3.4 query_params Struct Reference

struct for storing information about the query parameters

`#include <utils.h>`

Public Attributes

- char **column_names** [MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN]
- char **operator[MAX_COLUMNS_PER_TABLE]**
- char **value** [MAX_COLUMNS_PER_TABLE][MAX_VALUE_LEN]

3.4.1 Detailed Description

struct for storing information about the query parameters

Definition at line 222 of file utils.h.

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

- utils.h

3.5 record_struct Struct Reference

Public Attributes

- char **key** [MAX_KEY_LEN]
- int **metadata**
- record_struct * **next**
- column_struct **column**

3.5.1 Detailed Description

Definition at line 75 of file utils.h.

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

- utils.h

3.6 set_params Struct Reference

struct for storing information about the set parameters

`#include <utils.h>`

Public Attributes

- char **column_names** [MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN]
- char **value** [MAX_COLUMNS_PER_TABLE][MAX_VALUE_LEN]

3.6.1 Detailed Description

struct for storing information about the set parameters

Definition at line 242 of file utils.h.

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

- utils.h

3.7 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.

3.7.1 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

3.8 table Struct Reference

Public Attributes

- char **table_name** [MAX_TABLE_LEN]
- record_struct * **RecordStruct** [1000]

3.8.1 Detailed Description

Definition at line 85 of file utils.h.

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

- utils.h

3.9 table_list Struct Reference

Public Attributes

- table **Table** [100]

3.9.1 Detailed Description

Definition at line 92 of file utils.h.

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

- utils.h

3.10 table_params Struct Reference

struct for storing information about column names in tables in configuration file

`#include <utils.h>`

Public Attributes

- char **table_name** [MAX_TABLE_LEN]
- char **table_columns** [MAX_COLUMNS_PER_TABLE][MAX_COLNAME_LEN]
- int **column_types** [MAX_COLUMNS_PER_TABLE]

3.10.1 Detailed Description

struct for storing information about column names in tables in configuration file

Definition at line 113 of file utils.h.

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

- utils.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 <stdbool.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>
```

Macros

- #define **LOGGING** 0
- #define **THREECOLSTABLE** "threecols"
- #define **FOURCOLSTABLE** "fourcols"
- #define **SIXCOLSTABLE** "sixcols"
- #define **KEY1** "somekey1"
- #define **KEY2** "somekey2"
- #define **KEY3** "somekey3"

Functions

- int main (int argc, char *argv[])

  Start a client to interact with the storage server.

Variables

- FILE * file_ptr

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 41 of file client.c.

References MAX_KEY_LEN, MAX_RECORDS_PER_TABLE, and storage_record::value.

4.1.3 Variable Documentation

4.1.3.1 FILE∗ file_ptr

added struct RecordElement to store key, data, and pointer to next RecordElement

Definition at line 30 of file client.c.

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 <errno.h>
#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 <time.h>
#include <stdbool.h>
#include <pthread.h>
```

Classes

- struct _ThreadInfo

  Structure storing information for each thread.

Macros

- #define **LOGGING** 0
- #define MAX_LISTENQUEUELEN 20

  The maximum number of queued connections.

Typedefs

- typedef struct _ThreadInfo * **ThreadInfo**

Functions

- ThreadInfo getThreadInfo (void)

  Function accessing threadinfo struct.
- void releaseThread (ThreadInfo me)

  Function called when thread is about to finish – unless it is called, the ThreadInfo assigned to it is lost.
- int handle_command (FILE *file, int sock, char *cmd, table_list *List, struct config_params params, struct set_params sparams, struct query_params qparams)

  Process a command from the client.
- int main (int argc, char *argv[])

  Start the storage server.

Variables

- ThreadInfo **runtimeThreads** [MAX_CONNECTIONS]
- unsigned int **botRT** = 0
- unsigned int **topRT** = 0
- pthread_mutex_t **conditionMutex** = PTHREAD_MUTEX_INITIALIZER
- pthread_mutex_t **handle_cmd_lock** = PTHREAD_MUTEX_INITIALIZER
- pthread_cond_t **conditionCond** = PTHREAD_COND_INITIALIZER

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 ThreadInfo getThreadInfo (void)

Function accessing threadinfo struct.

Parameters

void	There're no parameter inputs Returns a pointer to the threadinfo struct

Definition at line 56 of file server.c.

References MAX_CONNECTIONS.

4.3.2.2 int handle_command (FILE * *file*, int *sock*, char * *cmd*, table_list * *List*, struct config_params *params*, struct set_params *sparams*, struct query_params *qparams*)

Process a command from the client.

Parameters

sock	The socket connected to the client.
cmd	The command received from the client.

Returns

Returns 0 on success, -1 otherwise.

Definition at line 118 of file server.c.

References ERR_AUTHENTICATION_FAILED, ERR_INVALID_PARAM, ERR_KEY_NOT_FOUND, ERR_TABLE_NOT_FOUND, ERR_UNKNOWN, logger(), config_params::pass_, query_parsing(), sendall(), set_parsing(), and config_params::username.

Referenced by main().

4.3.2.3 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 proccesses commands from clients.

Definition at line 394 of file server.c.

References config_params::concurrency, handle_command(), logger(), MAX_CMD_LEN, MAX_LISTENQUEUELEN, read_config(), recvline(), config_params::server_host, and config_params::server_port.

4.3.2.4 void releaseThread (ThreadInfo *me*)

Function called when thread is about to finish – unless it is called, the ThreadInfo assigned to it is lost.

Parameters

Threadinfo	the pointer to the threadinfo struct is passed in void There's no return

Definition at line 82 of file server.c.

References MAX_CONNECTIONS.

4.4 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.

4.4.1 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.

4.4.2 Function Documentation

4.4.2.1 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 the user.
password	the password of the user.
conn	the connection socket.

Returns

0 if successful, -1 if fail.

Connection is really just a socket file descriptor.

Send some data.

Definition at line 156 of file storage.c.

References ERR_AUTHENTICATION_FAILED, ERR_CONNECTION_FAIL, ERR_INVALID_PARAM, ERR_UNKNOWN, generate_encrypted_password(), MAX_CMD_LEN, MAX_ENC_PASSWORD_LEN, MAX_USERNAME_LEN, recvline(), and sendall().

4.4.2.2 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	the name of the host.
port	the port number.

Returns

void* sock pointer if successful, else returns NULL pointer.

Create a socket.

Get info about the server.

unknown error might happen due to a failed attempt to create a string with certain with the port number

unable to return a pointer to a linked list of addrinfo

Connect to the server.

unable to connect to server based on file descriptor of socket

Definition at line 82 of file storage.c.

References ERR_CONNECTION_FAIL, ERR_INVALID_PARAM, ERR_UNKNOWN, MAX_HOST_LEN, and MAX_PORT_LEN.

4.4.2.3 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. Cleanup

Definition at line 511 of file storage.c.

References ERR_CONNECTION_FAIL, and ERR_INVALID_PARAM.

4.4.2.4 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	is the particular table containing the record.
key	is the table[key] containing the linked list containing the record.
record	is the record node inside linked list.
conn	is the connection socket.

Returns

0 if successful, -1 if fail.

Connection is really just a socket file descriptor.

Send some data.

Definition at line 232 of file storage.c.

References ERR_CONNECTION_FAIL, ERR_INVALID_PARAM, ERR_KEY_NOT_FOUND, ERR_NOT_AUTHENTICATED, ERR_TABLE_NOT_FOUND, ERR_UNKNOWN, MAX_CMD_LEN, MAX_KEY_LEN, MAX_TABLE_LEN, storage_record::metadata, recvline(), sendall(), and storage_record::value.

4.4.2.5 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". Connection is really just a socket file descriptor.

Send some data.

Definition at line 409 of file storage.c.

References ERR_CONNECTION_FAIL, ERR_INVALID_PARAM, ERR_KEY_NOT_FOUND, ERR_NOT_AUTHENTICATED, ERR_TABLE_NOT_FOUND, ERR_UNKNOWN, MAX_CMD_LEN, MAX_TABLE_LEN, recvline(), and sendall().

4.4.2.6 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	is the particular table containing the record.
key	is the table[key] containing the linked list containing the record.
record	is the record node inside linked list.
conn	is the connection socket.

Returns

0 returns 0 if successful, -1 if fail.

Connection is really just a socket file descriptor.

Send some data.

Definition at line 325 of file storage.c.

References ERR_CONNECTION_FAIL, ERR_INVALID_PARAM, ERR_KEY_NOT_FOUND, ERR_NOT_AUTHENTICATED, ERR_TABLE_NOT_FOUND, ERR_TRANSACTION_ABORT, ERR_UNKNOWN, MAX_CMD_LEN, MAX_KEY_LEN, MAX_TABLE_LEN, storage_record::metadata, recvline(), sendall(), and storage_record::value.

4.5 utils.c File Reference

This file implements various utility functions that are 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 <errno.h>
#include "storage.h"
```

Functions

- record_struct * **find_record** (char *table, char *key, table_list *list)
- bool **is_number** (char *value)
- int keymaker (const char *key)

  This is a function to generate a key from a string.
- bool **find_table** (char *table, table_list *table_ptr)
- int **record_get** (char *table, char *key, table_list *list, char *append, int *metadata_ptr)
- int **delete_record** (table_list *list, char *table, char *key)
- int **create_record** (int option, int i, char *key, char *value, table_list *List, config_params params, int metadata)
- int **insert_key** (const char *table, const char *key, const char *value, table_list *table_ptr, const config_params params, int metadata)
- int **insert_table** (config_params params, table_list *table_ptr)
- int **record_query** (char *tableName, char *predicates, int max_keys, char *keys_found, table_list *list)
- bool is_alpha (char str_ptr)

  Function that determines whether a character is alphanumeric character or not by using ASCII values.
- 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)

  Reads the stream one byte at a time.
- int process_config_line (char *line, struct config_params *params)

  Read and load configuration parameters.
- 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.
- char * query_parsing (char *unparsed_, struct query_params *qparams)

  This is a function that parses the query command from client side to be used in server side.

- char * set_parsing (char *unparsed_, struct set_params *sparams)

  This is a function that parses the set command from client side to be used in server side.

4.5.1 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.

4.5.2 Function Documentation

4.5.2.1 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 1223 of file utils.c.

References DEFAULT_CRYPT_SALT.

Referenced by storage_auth().

4.5.2.2 bool is_alpha (char *str_ptr*)

Function that determines whether a character is alphanumeric character or not by using ASCII values.

Parameters

str_ptr	The character that's being evaluated. If the character is alphanumeric, return true. If not, return false.

Definition at line 546 of file utils.c.

4.5.2.3 int keymaker (const char * *key*)

This is a function to generate a key from a string.

hashing function

Parameters

key	String to be hashed into an integer key

Returns

Returns the integer key

Address to the record

Get a certain number from multiplication of ascii char

Makes a key from the result of both above

Definition at line 96 of file utils.c.

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

Generates a log message.

Parameters

file	The output stream
message	Message to log.

Definition at line 1208 of file utils.c.

Referenced by handle_command(), and main().

4.5.2.5 int process_config_line (char * *line,* struct config_params * *params*)

Read and load configuration parameters.

Parameters

line	Line of input read from file.
params	The structure where config parameters are loaded.

Returns

Return 0 on success, -1 otherwise.

Commented lines ignored.

Extract config parameter name and value.

Line wasn't as expected.

Process this line.

if duplicate table name already exists

if duplicate column name already exists

if duplicate column name already exists

else if (strcmp(name, "data_directory") == 0) { strncpy(params->data_directory, value, sizeof params->data_directory); }

Ignore unknown config parameters.

Definition at line 623 of file utils.c.

References config_params::concurrency, MAX_COLNAME_LEN, MAX_COLUMNS_PER_TABLE, MAX_CONFIG_LINE_LEN, MAX_TABLE_LEN, MAX_TABLES, config_params::pass_, config_params::server_host, config_params-

::server_port, and config_params::username.

Referenced by read_config().

4.5.2.6 char* query_parsing (char * *unparsed_*, struct query_params * *qparams*)

This is a function that parses the query command from client side to be used in server side.

Parameters

*unparsed_*	pointer to the unparsed string
query_params	*qparams pointer to the struct where query parameters are stored

Returns

pointer to the parsed string

First predicate parsing ends here

Next predicates parsing start here

predicate parsing ends here

Next predicates parsing end here

Putting it together in a string

Definition at line 1238 of file utils.c.

References MAX_COLNAME_LEN, MAX_COLUMNS_PER_TABLE, and MAX_VALUE_LEN.

Referenced by handle_command().

4.5.2.7 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.
params	The structure where config parameters are loaded.

Returns

Return 0 on success, -1 otherwise.

Open file for reading.

Process the config file.

Read a line from the file.

right size, proceed to process line

not the right size

Definition at line 1093 of file utils.c.

References config_params::concurrency, ERR_UNKNOWN, MAX_CONFIG_LINE_LEN, config_params::pass_, process_config_line(), config_params::server_host, config_params::server_port, and config_params::username.

Referenced by main().

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

Reads the stream one byte at a time.

Receive an entire line from a socket.

Parameters

sock	Connection socket
buf	String buffer
buflen	Length of buffer

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.

Return status.

Read one byte from scoket.

recv() was not successful, so stop.

Found end of line, so stop.

Keep going.

add null terminator in case it's not already there.

Definition at line 588 of file utils.c.

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

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

Keep sending the contents of the buffer until complete.

Parameters

sock	Connection socket
buf	String buffer
len	Length of buffer

Returns

Return 0 on success, -1 otherwise.

The parameters mimic the send() function. send() was not successful, so stop.

Definition at line 563 of file utils.c.

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

4.5.2.10 char* set_parsing (char * *unparsed_*, struct **set_params** * *sparams*)

This is a function that parses the set command from client side to be used in server side.

Parameters

*unparsed_*	pointer to the unparsed string
set_params	∗sparams pointer to the struct where set parameters are stored

Returns

pointer to the parsed string

First predicate parsing ends here

Next predicates parsing start here

predicate parsing ends here

Next predicates parsing end here

Putting it together in a string

Definition at line 1540 of file utils.c.

References MAX_COLNAME_LEN, MAX_COLUMNS_PER_TABLE, and MAX_VALUE_LEN.

Referenced by handle_command().

4.6 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 <stdbool.h>
#include <pthread.h>
```

Classes

- struct column_struct
- struct record_struct
- struct table
- struct table_list
- struct table_params

  struct for storing information about column names in tables in configuration file
- struct config_params

  struct for storing information about the current configuration file
- struct query_params

  struct for storing information about the query parameters
- struct set_params

  struct for storing information about the set parameters

Macros

- #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.

Typedefs

- typedef struct column_struct **column_struct**
- typedef struct record_struct **record_struct**
- typedef struct table **table**
- typedef struct table_list **table_list**
- typedef struct table_params **table_params**
- typedef struct config_params **config_params**

Functions

- int keymaker (const char *index)

  hashing function
- bool is_alpha (char str_ptr)

  Function that determines whether a character is alphanumeric character or not by using ASCII values.
- 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.
- char * query_parsing (char *unparsed_, struct query_params *qparams)

  This is a function that parses the query command from client side to be used in server side.
- char * set_parsing (char *unparsed_, struct set_params *sparams)

  This is a function that parses the set command from client side to be used in server side.
- int **record_query** (char *tableName, char *predicates, int max_keys, char *keys_found, table_list *list)
- int **insert_table** (config_params params, table_list *table_ptr)
- int **insert_key** (const char *table, const char *key, const char *value, table_list *table_ptr, config_params params, int metadata)
- int **create_record** (int option, int i, char *key, char *value, table_list *List, config_params params, int metadata)
- int **delete_record** (table_list *list, char *table, char *key)
- int **record_get** (char *table, char *key, table_list *list, char *append, int *metadata)
- bool **find_table** (char *table, table_list *table_ptr)
- record_struct * **find_record** (char *table, char *key, table_list *list)

Variables

- FILE * file_ptr

4.6.1 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.

4.6.2 Macro Definition Documentation

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

A macro to output debug information.

It is only enabled in debug builds.

Definition at line 54 of file utils.h.

4.6.2.2 #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 44 of file utils.h.

4.6.3 Function Documentation

4.6.3.1 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 1223 of file utils.c.

References DEFAULT_CRYPT_SALT.

Referenced by storage_auth().

4.6.3.2 bool is_alpha (char str_ptr)

Function that determines whether a character is alphanumeric character or not by using ASCII values.

Parameters

str_ptr	The character that's being evaluated. If the character is alphanumeric, return true. If not, return false.

Definition at line 546 of file utils.c.

4.6.3.3 int keymaker (const char * *key*)

hashing function

Parameters

index	The key to be inserted

Returns

Returns hashed int key

hashing function

Parameters

key	String to be hashed into an integer key

Returns

Returns the integer key

Address to the record

Get a certain number from multiplication of ascii char

Makes a key from the result of both above

Definition at line 96 of file utils.c.

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

Generates a log message.

Parameters

file	The output stream
message	Message to log.

Definition at line 1208 of file utils.c.

Referenced by handle_command(), and main().

4.6.3.5 char* query_parsing (char * *unparsed_,* struct query_params * *qparams*)

This is a function that parses the query command from client side to be used in server side.

Parameters

*unparsed_*	pointer to the unparsed string
query_params	∗qparams pointer to the struct where query parameters are stored

Returns

pointer to the parsed string

First predicate parsing ends here

Next predicates parsing start here

predicate parsing ends here

Next predicates parsing end here

Putting it together in a string

Definition at line 1238 of file utils.c.

References MAX_COLNAME_LEN, MAX_COLUMNS_PER_TABLE, and MAX_VALUE_LEN.

Referenced by handle_command().

4.6.3.6 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.
params	The structure where config parameters are loaded.

Returns

Return 0 on success, -1 otherwise.

Open file for reading.

Process the config file.

Read a line from the file.

right size, proceed to process line

not the right size

Definition at line 1093 of file utils.c.

References config_params::concurrency, ERR_UNKNOWN, MAX_CONFIG_LINE_LEN, config_params::pass_, process_config_line(), config_params::server_host, config_params::server_port, and config_params::username.

Referenced by main().

4.6.3.7 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.

Receive an entire line from a socket.

Parameters

sock	Connection socket
buf	String buffer
buflen	Length of buffer

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.

Return status.

Read one byte from scoket.

recv() was not successful, so stop.

Found end of line, so stop.

Keep going.

add null terminator in case it's not already there.

Definition at line 588 of file utils.c.

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

4.6.3.8 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.

Parameters

sock	Connection socket
buf	String buffer
len	Length of buffer

Returns

Return 0 on success, -1 otherwise.

The parameters mimic the send() function. send() was not successful, so stop.

Definition at line 563 of file utils.c.

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

4.6.3.9 char* set_parsing (char * *unparsed_*, struct set_params * *sparams*)

This is a function that parses the set command from client side to be used in server side.

Parameters

*unparsed_*	pointer to the unparsed string
set_params	*sparams pointer to the struct where set parameters are stored

Returns

pointer to the parsed string

First predicate parsing ends here

Next predicates parsing start here

predicate parsing ends here

Next predicates parsing end here

Putting it together in a string

Definition at line 1540 of file utils.c.

References MAX_COLNAME_LEN, MAX_COLUMNS_PER_TABLE, and MAX_VALUE_LEN.

Referenced by handle_command().

4.6.4 Variable Documentation

4.6.4.1 FILE* file_ptr

added struct RecordElement to store key, data, and pointer to next RecordElement

Definition at line 30 of file client.c.

Index