ECE297 Storage Server

0.2

Generated by Doxygen 1.7.1

Tue Mar 26 2013 22:10:55

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.

`#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_names [MAX_TABLES][MAX_TABLE_LEN]

  Heindrik: The directory where table names are stored.

- int **num_tables**

3.1.1 Detailed Description

A struct to store config parameters.

Definition at line 52 of file utils.h.

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

- utils.h

3.2 configuration Struct Reference

Public Attributes

- char * **host**
- char * **username**
- char * **password**
- int **port**
- int **num_tables**
- char **all_table_names** [MAX_TABLES][MAX_TABLE_LEN]
- char **set_values** [20][20]
- int **numValues**
- char **predicates** [100][100]
- int **totPredicates**
- struct table * **tlist**

3.2.1 Detailed Description

Definition at line 85 of file utils.h.

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

- utils.h

3.3 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.

3.3.1 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

3.4 table Struct Reference

Public Attributes

- int **numkeys**
- int **numCols**
- int **row_index**
- char * **table_name**
- char **array_config** [1024][MAX_COLNAME_LEN]
- char **array_empty** [4096][MAX_COLNAME_LEN]
- char **array_keys** [MAX_RECORDS_PER_TABLE][MAX_KEY_LEN]
- struct table * **next**

3.4.1 Detailed Description

Definition at line 74 of file utils.h.

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

- utils.h

3.5 TreeDB Struct Reference

File: struct TreeDB.h Author: MatthewMarji.

```
#include <TreeDB.h>
```

Public Attributes

- struct TableNode * **root**

3.5.1 Detailed Description

File: struct TreeDB.h Author: MatthewMarji. Created on February 16, 2013, 12:35 AM

Definition at line 18 of file TreeDB.h.

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

- TreeDB.h

3.6 TreeEntry Struct Reference

Public Attributes

- char * **name**
- char * **value**
- struct TreeDB * **tree**

3.6.1 Detailed Description

Definition at line 14 of file TreeEntry.h.

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

- TreeEntry.h

3.7 TreeNode Struct Reference

Public Attributes

- struct TreeEntry * **entryPtr**

- struct TreeNode * **left**
- struct TreeNode * **right**

3.7.1 Detailed Description

Definition at line 17 of file TreeNode.h.

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

- TreeNode.h

3.8 yy_buffer_state Struct Reference

Public Attributes

- FILE * **yy_input_file**
- char * **yy_ch_buf**
- char * **yy_buf_pos**
- yy_size_t **yy_buf_size**
- int **yy_n_chars**
- int **yy_is_our_buffer**
- int **yy_is_interactive**
- int **yy_at_bol**
- int yy_bs_lineno
- int yy_bs_column
- int **yy_fill_buffer**
- int **yy_buffer_status**

3.8.1 Detailed Description

Definition at line 216 of file lex.qq.c.

3.8.2 Member Data Documentation

3.8.2.1 int yy_buffer_state::yy_bs_column

The column count.

Definition at line 253 of file lex.qq.c.

3.8.2.2 int yy_buffer_state::yy_bs_lineno

The line count.

Definition at line 252 of file lex.qq.c.

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

- lex.qq.c
- lex.ss.c
- lex.yy.c

3.9 yy_trans_info Struct Reference

Public Attributes

- flex_int32_t **yy_verify**
- flex_int32_t **yy_nxt**

3.9.1 Detailed Description

Definition at line 394 of file lex.qq.c.

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

- lex.qq.c
- lex.ss.c
- lex.yy.c

3.10 yyalloc Union Reference

Public Attributes

- yytype_int16 **yyss_alloc**
- YYSTYPE **yyvs_alloc**

3.10.1 Detailed Description

Definition at line 325 of file config_parser.tab.c.

The documentation for this union was generated from the following files:

- config_parser.tab.c
- query.tab.c
- set.tab.c

3.11 YYSTYPE Union Reference

Public Attributes

- char * **sval**
- int **pval**
- int **ival**

3.11.1 Detailed Description

Definition at line 140 of file config_parser.tab.c.

The documentation for this union was generated from the following files:

- config_parser.tab.c
- config_parser.tab.h
- query.tab.c
- query.tab.h
- set.tab.c
- set.tab.h

Chapter 4

File Documentation

4.1 autoshell.c File Reference

This file implements a "very" simple sample client.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include <errno.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 * **fpCLIENT**

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 autoshell.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 31 of file autoshell.c.

References 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 client.c File Reference

This file implements a "very" simple sample client.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include <errno.h>
#include "storage.h"
#include "utils.h"
```

Functions

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

  Start a client to interact with the storage server.

Variables

- FILE * **fpCLIENT**

4.2.1 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.

4.2.2 Function Documentation

4.2.2.1 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_HOST_LEN, MAX_KEY_LEN, MAX_TABLE_LEN, MAX_USERNAME_LEN, storage_auth(), storage_connect(), storage_disconnect(), storage_get(), storage_query(), storage_set(), and storage_record::value.

4.3 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[ ])

4.3.1 Detailed Description

This program implements a password encryptor.

Definition in file encrypt_passwd.c.

4.4 server.c File Reference

This file implements the storage server.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <assert.h>
#include <signal.h>
#include <time.h>
#include <errno.h>
#include "utils.h"
#include "TreeDB.h"
#include "TreeNode.h"
#include "TreeEntry.h"
```

Defines

- #define **LOGGING** 1
- #define MAX_LISTENQUEUELEN 20

  The maximum number of queued connections.

- #define **STRING_LENGTH** 20

Functions

- int **handle_command** (int sock, char *cmd, struct TreeDB *tree, config_params params, int lex_status)
- int main (int argc, char *argv[ ])

  Start the storage server.

Variables

- struct configuration * **c**
- struct table * **tl**
- struct table * **t**
- FILE * **fpSERVER**
- int **error_status**

4.4.1 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.

4.4.2 Function Documentation

4.4.2.1 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 1046 of file server.c.

References createTable(), currentDateTime(), logger(), MAX_CMD_LEN, MAX_ENC_PASSWORD_LEN, MAX_KEY_LEN, MAX_LISTENQUEUELEN, MAX_TABLE_LEN, MAX_USERNAME_LEN, MAX_VALUE_LEN, config_params::password, recvline(), config_params::server_host, config_params::server_port, timeval_subtract(), and config_params::username.

4.5 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 <errno.h>
#include "storage.h"
#include "client.h"
#include "utils.h"
#include <sys/time.h>
```

Defines

- #define **LOGGING** 1

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 the authentication function, this function will send the client information to the server and verify that information with the deafult.conf file. Once the information has been verified this function will return a code depending on the status of the verification. if it returned 0 then the authentication is successful. If another number is returned however, then an error occured. The type of error is determined by the code that is returned.

- 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_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)

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

Variables

- int **AUTH_STATUS** = 1
- char **holder** [100]
- FILE * **fpCLIENT** = NULL

4.5.1 Detailed Description

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

Definition in file storage.c.

4.5.2 Function Documentation

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

This is the authentication function, this function will send the client information to the server and verify that information with the deafult.conf file. Once the information has been verified this function will return a code depending on the status of the verification. if it returned 0 then the authentication is successful. If another number is returned however, then an error occured. The type of error is determined by the code that is returned.

Authenticate the client's connection to the server.

Definition at line 105 of file storage.c.

References ERR_AUTHENTICATION_FAILED, generate_encrypted_password(), 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

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.

Client connects! REASONING: We will have to ensure that the user correctly enters the required hostname and password. we will LOG what the user enters for the hostname and password, as well as what time they connect. NOTE: If this log does not get written, the client has likely not entered a valid hostname and/or password LOGGER 1! If connection to server not made, return NULL, ELSE if connection made, return socket value (integer).

Definition at line 31 of file storage.c.

References currentDateTime(), and logger().

Referenced by main().

4.5.2.3 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 514 of file storage.c.

References currentDateTime(), and logger().

Referenced by main().

4.5.2.4 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 174 of file storage.c.

References ERR_KEY_NOT_FOUND, ERR_TABLE_NOT_FOUND, logger(), recvline(), sendall(), valid_string_check(), 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

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".

Definition at line 368 of file storage.c.

References ERR_INVALID_PARAM, ERR_TABLE_NOT_FOUND, logger(), MAX_KEY_LEN, recvline(), sendall(), and valid_string_check().

Referenced by main().

4.5.2.6 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 263 of file storage.c.

References currentDateTime(), logger(), recvline(), sendall(), valid_string_check(), and storage_record::value.

Referenced by main().

4.6 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.

4.6.1 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.

4.6.2 Function Documentation

4.6.2.1 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.

Definition at line 105 of file storage.c.

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

Referenced by main().

4.6.2.2 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.

Client connects! REASONING: We will have to ensure that the user correctly enters the required hostname and password. we will LOG what the user enters for the hostname and password, as well as what time they connect. NOTE: If this log does not get written, the client has likely not entered a valid hostname and/or password LOGGER 1! If connection to server not made, return NULL, ELSE if connection made, return socket value (integer).

Definition at line 31 of file storage.c.

References currentDateTime(), and logger().

Referenced by main().

4.6.2.3 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.

Definition at line 514 of file storage.c.

References currentDateTime(), and logger().

Referenced by main().

4.6.2.4 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.

Definition at line 174 of file storage.c.

References ERR_KEY_NOT_FOUND, ERR_TABLE_NOT_FOUND, logger(), recv-line(), sendall(), valid_string_check(), and storage_record::value.

Referenced by main().

4.6.2.5 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".

Definition at line 368 of file storage.c.

References ERR_INVALID_PARAM, ERR_TABLE_NOT_FOUND, logger(), MAX_KEY_LEN, recvline(), sendall(), and valid_string_check().

Referenced by main().

4.6.2.6 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.

Definition at line 263 of file storage.c.

References currentDateTime(), logger(), recvline(), sendall(), valid_string_check(), and storage_record::value.

Referenced by main().

4.7 TreeEntry.c File Reference

GET and SET OPERATION are implemented here.

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

```
#include "TreeEntry.h"
#include "TreeDB.h"
#include "storage.h"
```

Functions

- struct TreeEntry * createTable (char *name)

  EVERY TABLE HAS AN ENTRY, so we create a TABLE with a key, value, and table-name.

- void deleteTable (struct TreeEntry *table)

  SAFELY REMOVE THE TABLE.

- void setTableName (struct TreeEntry *table, char *name)

  TABLES ARE DEFINED BY NAMES. HERE WE WILL ASSIGN THE NAME TO THE TABLE.

- char * getTableName (struct TreeEntry *table)
- char * getEntryValue (struct TreeEntry *table)
- void setEntryValue (struct TreeEntry *table, char *_value)

  sets entry value to _value

- void printTable (struct TreeEntry *table)

  prints the name of the table

4.7.1 Detailed Description

GET and SET OPERATION are implemented here.

Definition in file TreeEntry.c.

4.7.2 Function Documentation

4.7.2.1 char* getEntryValue (struct TreeEntry * *table*)

Returns

returns string to entry value

Definition at line 88 of file TreeEntry.c.

4.7.2.2 char* getTableName (struct TreeEntry * *table*)

Returns

RETURN THE TABLE NAME

Definition at line 76 of file TreeEntry.c.

Referenced by findTable(), insertTable(), and printTable().

4.8 TreeNode.c File Reference

In this file we will create the necessary functions to insert, delete, get and set the entries for the tables. Every TableNode has the following: left pointer right pointer struct entry -> has a key and value. place the Node in the BST based on the entry->key value.

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include "TreeNode.h"
#include "TreeDB.h"
```

Functions

- struct TreeNode * createTreeNode (struct TreeEntry *entryPtr)
  Create an entry node for storage in the appropriate table.
- void **deleteTreeNode** (struct TreeNode *node)
- void setLeft (struct TreeNode *current, struct TreeNode *newLeft)
  sets the left child of the struct TreeNode.
- void setRight (struct TreeNode *current, struct TreeNode *newRight)
  sets the right child of the struct TreeNode
- struct TreeNode * getLeft (struct TreeNode *current)
  gets the left child of the struct TreeNode
- struct TreeNode * getRight (struct TreeNode *current)
  gets the right child of the struct TreeNode
- struct TreeEntry * getEntry (struct TreeNode *node)

returns a pointer to the DBentry the struct TreeNode contains.

- struct TreeEntry * findTable (struct TreeNode *node, char *name)

  Search for an entry by comparing a string to the current keys in the BST. If the entry is found, we will return the structure so that the elements can be accessed as necessary.

- bool insertTable (struct TreeNode *node, struct TreeEntry *newEntry)

  Inserts, if it already exists returns false, otherwise true.

- struct TreeNode * **removeTable** (struct TreeNode *node, char *name, struct TreeNode **check)
- struct TreeNode * getLargest (struct TreeNode *node)

  In the case where we have to delete an entry that has two entries (to the left, and to the right) We must find the MAX entry from the left subtree and replace it with this.

- void **printNodes** (struct TreeNode *node)

4.8.1 Detailed Description

In this file we will create the necessary functions to insert, delete, get and set the entries for the tables. Every TableNode has the following: left pointer right pointer struct entry -> has a key and value. place the Node in the BST based on the entry->key value.

Definition in file TreeNode.c.

4.8.2 Function Documentation

4.8.2.1 struct TreeNode* getLargest (struct TreeNode * *node*) [read]

In the case where we have to delete an entry that has two entries (to the left, and to the right) We must find the MAX entry from the left subtree and replace it with this.

Returns

Returning the largest element

Definition at line 249 of file TreeNode.c.

4.9 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 <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <unistd.h>
#include <time.h>
#include "utils.h"
#include <math.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, config_params *params, int s_table)

  Parse and process a line in the config file.

- int read_config (const char *config_file, config_params *params)

  Read and load configuration parameters.

- void logger (FILE *file, char *message, int LOGGING)

  Generates a log message.

- char * currentDateTime ()

  Default two character salt used for password encryption.

- char * generate_encrypted_password (const char *passwd, const char *salt)

  Generates an encrypted password string using salt CRYPT_SALT.

- int valid_string_check (char *input, int value)

  This "valid_string_check" function will take in a string and check whether this string contains the proper ASCII values from A-Z and 0-9. If it is then the function returns a zero if not then the function returns another value.

- char * add_delimiter (char *input)

  this function will replace all the spaces within a string into a '#' for storing purposes in the value.

- char * undo_delimiter (char *input)

  this function will replace all the '#' within a string into a ' ' for retrieving purposes in the value.

- int timeval_subtract (struct timeval *result, struct timeval *t2, struct timeval *t1)

  The following two functions below are used to time the execution time of the server, and the storage. "timeval_subtract" subtracts final and the initial value to get the elapsed time. "timeval_print" will print out the value of "timeval_subtract".

- int type_identifier (char string[800])

  The type_identifier function will receive a string and check whether it is all numbers or not. if it is all integers we return 0 identifying that the string sent is an integer else we send 0 which identifies that the string sent is a char value.

- int **size_get** (char string[800])

4.9.1 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.

4.9.2 Function Documentation

4.9.2.1 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 207 of file utils.c.

Referenced by storage_auth().

4.9.2.2 void logger (FILE * *file*, char * *message*, int *LOGGER*)

Generates a log message.

Parameters

file The output stream

message Message to log.

Definition at line 180 of file utils.c.

Referenced by main(), storage_connect(), storage_disconnect(), storage_get(), storage_query(), and storage_set().

4.9.2.3 int read_config (const char * *config_file*, 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.

Definition at line 126 of file utils.c.

References process_config_line().

4.9.2.4 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 40 of file utils.c.

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

4.9.2.5 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 20 of file utils.c.

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

4.10 utils.h File Reference

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

```
#include <errno.h>
#include <stdio.h>
#include "storage.h"
#include <sys/time.h>
```

Classes

- struct config_params

  A struct to store config parameters.

- struct table
- struct configuration

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"

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, config_params *params)

  Read and load configuration parameters.

- void logger (FILE *file, char *message, int LOGGER)

  Generates a log message.

- char * currentDateTime ()

  Default two character salt used for password encryption.

- char * generate_encrypted_password (const char *passwd, const char *salt)

  Generates an encrypted password string using salt CRYPT_SALT.

- int valid_string_check (char *input, int value)

  This "valid_string_check" function will take in a string and check whether this string contains the proper ASCII values from A-Z and 0-9. If it is then the function returns a zero if not then the function returns another value.

- char * add_delimiter (char *input)

  this function will replace all the spaces within a string into a '#' for storing purposes in the value.

- char * undo_delimiter (char *input)

  this function will replace all the '#' within a string into a ' ' for retrieving purposes in the value.

- int timeval_subtract (struct timeval *result, struct timeval *t2, struct timeval *t1)

  The following two functions below are used to time the execution time of the server, and the storage. "timeval_subtract" subtracts final and the initial value to get the elapsed time. "timeval_print" will print out the value of "timeval_subtract".

- void **timeval_print** (struct timeval *tv)

- int type_identifier (char string[800])

  The type_identifier function will receive a string and check whether it is all numbers or not. if it is all integers we return 0 identifying that the string sent is an integer else we send 0 which identifies that the string sent is a char value.

- int **size_get** (char string[800])

Variables

- struct configuration * **c**
- struct table * **tl**
- struct table * **t**

4.10.1 Detailed Description

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

Definition in file utils.h.

4.10.2 Define Documentation

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

A macro to output debug information.

It is only enabled in debug builds.

Definition at line 46 of file utils.h.

4.10.2.2 #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 34 of file utils.h.

4.10.3 Function Documentation

4.10.3.1 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 207 of file utils.c.

Referenced by storage_auth().

4.10.3.2 void logger (FILE * *file*, char * *message*, int *LOGGER*)

Generates a log message.

Parameters

file The output stream

message Message to log.

Definition at line 180 of file utils.c.

Referenced by main(), storage_connect(), storage_disconnect(), storage_get(), storage_query(), and storage_set().

4.10.3.3 int read_config (const char * *config_file*, 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.

Definition at line 126 of file utils.c.

References process_config_line().

4.10.3.4 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 40 of file utils.c.

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

4.10.3.5 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 20 of file utils.c.

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

Index