libtdd

Generated by Doxygen 1.8.14

Contents

1 Main Page

This library provides a simple framework for defining, organizing, and running unit tests in C. It is inspired by the golang `testing` pkg.

Some minimal boilerplate is required to write a decent test suite for a software package, but this is straight-forward and can be quickly written in a main() function.

In a typical testing binary, the main() function should initialize tests and create any resource files, etc. that might be expected by tests in the suite.

1.1 Example usage of this library:

```
#include <stdlib.h>
#include "tdd.h"

static void* error_func(test_t* t) {
    test_error(t, "oops!");
    return NULL;
}

static void* fail_func(test_t* t) {
    test_fail(t, "badness!");
    return NULL;
}

int main(int argc, char* argv[]) {
    // create test suite
    suite_t* s = suite_new();

    // initalize tests
    // variadic func; add as many tests as needed
    suite_add(s, 2, tdd_testfn_new(&error_func, "error", NULL),
              tdd_testfn_new(&fail_func, "fail", NULL));

    // run the test suite
    suite_run(s);
    stats = suite_stats(s);
    suite_del(s);
    ...

    return stats.n_error
}
```

1.2 Test API

Test functions must always be defined by one of the following signatures:

- `void* test_func(test_t* t);` for regular tests
- `void* bench_func(test_t* t);` for benchmarking tests

These functions are added to a test suite using either the `suite_add(...)` or `suite_addtest(testfn_t*)` API calls.

1.3 Benchmarking

Test functions may be specially marked as time-sensitive, as in where performance is paramount. Tests may be marked for benchmarking by adding them to the suite with a name prefixed by `bench_`. For all benchmarked functions A timer will be started when the test runs, and stopped when the test finishes. A report of the runtime is printed after the test finishes.

For example:

```
suite_addtest(tdd_testfn_new(&bench_func, "bench_func", "time sensitive test"));
```

1.4 Notes

This library is multithreaded using POSIX `pthread`s. As such, any binaries built using this library must be compiled with either `gcc` or `clang`'s `-pthread` option.

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

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

4 Data Structure Documentation

4.1 suite_stats_t Struct Reference

`#include <tdd.h>`

Data Fields

- tdd_result_t ** tests_run
- int n_tests
- int n_error
- int n_fail
- int n_ran
- double success_rate
- bool fatal_failures

4.1.1 Detailed Description

Stats structure detailing results of test suite.

4.1.2 Field Documentation

4.1.2.1 fatal_failures

`bool suite_stats_t::fatal_failures`

`fatal_failures` is an indication that the suite ran with fatal failures enabled.

4.1.2.2 n_error

```
int suite_stats_t::n_error
```

`n_error` is the total number of errors in the suite.

4.1.2.3 n_fail

```
int suite_stats_t::n_fail
```

`n_fail` is the total number of failures in the suite.

4.1.2.4 n_ran

```
int suite_stats_t::n_ran
```

`n_ran` is the total number of tests that ran in the suite. If this count differs from `n_tests`, then some tests were skipped.

4.1.2.5 n_tests

```
int suite_stats_t::n_tests
```

`n_tests` is the total number of tests in the suite.

4.1.2.6 success_rate

```
double suite_stats_t::success_rate
```

`success_rate` is the percent rate of successful tests in the suite.

4.1.2.7 tests_run

```
tdd_result_t** suite_stats_t::tests_run
```

`test_run` is an array of tdd_result_t containing the results of the tests that ran in the suite.

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

- include/tdd.h

4.2 suite_t Struct Reference

```
#include <tdd.h>
```

Data Fields

- bool finished
- int n_tests
- int n_segv
- int test_index
- testfn_t ** tests
- test_t ** results
- FILE * outfile
- bool quiet

4.2.1 Detailed Description

Testing suite. Contains all tests, current runtime state, and the results of each test. May be used to contruct a suite_stats_t after running.

4.2.2 Field Documentation

4.2.2.1 finished

```
bool suite_t::finished
```

`finished` is a boolean flag specifying if all tests have run. If this flag is not set by the time the suite has completed all tests, then it aborted testing with a fatal failure from one of the tests in the suite.

4.2.2.2 n_segv

```
int suite_t::n_segv
```

`n_segv` is the number of segmentation faults that were caught.

4.2.2.3 n_tests

```
int suite_t::n_tests
```

`n_tests` is the number of tests in the suite.

4.2.2.4 outfile

```
FILE* suite_t::outfile
```

`outfile` is a FILE pointer which is where the results of the test will be printed. This is stdout by default, but may be changed manually after the suite is initialized and before it is run.

4.2.2.5 quiet

```
bool suite_t::quiet
```

`quiet` is a boolean flag indicating that results should not be printed as the test suite runs; reporting can instead be done through creating a stats structure after the suite finishes.

4.2.2.6 results

```
test_t** suite_t::results
```

`results` is an array of `test_t*` that details testing results for each element in tests.

4.2.2.7 test_index

```
int suite_t::test_index
```

`test_index` is the index of the current test.

4.2.2.8 tests

```
testfn_t** suite_t::tests
```

`tests` is an array of testfn_t* that make up the suite.

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

- include/tdd.h

4.3 tdd_result_t Struct Reference

```
#include <tdd.h>
```

Data Fields

- char * name
- bool ok

4.3.1 Detailed Description

Test result. This structure holds the name of a test which ran, and indicates if the test passed.

4.3.2 Field Documentation

4.3.2.1 name

`char* tdd_result_t::name`

`name` is the name of the test that produced this result.

4.3.2.2 ok

`bool tdd_result_t::ok`

`ok` indicates if the test that produced this result was successful.

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

- include/tdd.h

4.4 test_t Struct Reference

`#include <tdd.h>`

Data Fields

- const char * name
- bool failed
- int err
- char * fail_msg
- char ** err_msg
- struct timespec * start
- struct timespec * end
- struct timespec * failed_at
- struct timespec * error_at
- void(* fail)(struct test_t *t, char *msg)
- void(* error)(struct test_t *t, char *msg)
- void(* done)(struct test_t *t)

4.4.1 Detailed Description

Testing structure. This structure is the only parameter in all testing functions. If at any point during a testing function, unexpected bevahiour occurs or the test downright fails, you should call `test_error(t)` and `test_fail(t)` respectively.

4.4.2 Field Documentation

4.4.2.1 done

```
void(* test_t::done) (struct test_t *t)
```

done() marks the test as finished

4.4.2.2 end

```
struct timespec* test_t::end
```

end is the timestamp at which the test was marked as done. Heap allocated.

4.4.2.3 err

```
int test_t::err
```

err is a integer flag specifying the number of errors the current test has encountered.

4.4.2.4 err_msg

```
char** test_t::err_msg
```

err_msg is an array of character strings that is appended to on each call to test_error(). Each string in this array corresponds to the reason for errors in order of occurance. Each string is Heap allocated.

4.4.2.5 error

```
void(* test_t::error) (struct test_t *t, char *msg)
```

error() marks the test as having encountered an error with a message explaining the reason for the error

4.4.2.6 error_at

```
struct timespec* test_t::error_at
```

error_at is the timestamp at which the test last encountered an error. Heap allocated.

4.4.2.7 fail

```
void(* test_t::fail) (struct test_t *t, char *msg)
```

fail() marks the test as failed with a message explaining the reason for failure.

4.4.2.8 fail_msg

```
char* test_t::fail_msg
```

fail_msg is a message that is by test_fail() indicating the reason for test failure. Heap allocated.

4.4.2.9 failed

```
bool test_t::failed
```

`failed` is a boolean flag specifying if the current test has failed.

4.4.2.10 failed_at

```
struct timespec* test_t::failed_at
```

`failed_at` is the timestamp at which the test last encountered a failure. Heap allocated.

4.4.2.11 name

```
const char* test_t::name
```

`name` is a character string that describes the test result.

4.4.2.12 start

```
struct timespec* test_t::start
```

`start` is the timestamp at which the test was started. Heap allocated.

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

- include/tdd.h

4.5 testfn_s Struct Reference

```
#include <tdd.h>
```

Data Fields

- char * name
- char * desc
- void *(* fn)(void *t)

4.5.1 Detailed Description

Testing function.

4.5.2 Field Documentation

4.5.2.1 desc

```
char* testfn_s::desc
```

`desc` is a character string description for a test. It should be a humanly readable explanation of what the test is performing. Optional field in constructor.

4.5.2.2 fn

```
void*(* testfn_s::fn) (void *t)
```

`fn` is a pointer to a test function.

Parameters

t	- pointer to a `test_t` structure to capture context of test results

4.5.2.3 name

```
char* testfn_s::name
```

`name` is a character string identifier for a test. It is usually just the name of the function itself, such as "test_func".

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

- include/tdd.h

5 File Documentation

5.1 include/strutil.h File Reference

String format and manipulation functions and macros for libtdd.

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

Macros

- #define __INDENT(f, n) fprintf(f, "%.*s", n, " ")

5.1.1 Detailed Description

String format and manipulation functions and macros for libtdd.

Author

Keefer Rourke <mail@krourke.org

Date

08 Apr 2018

5.1.2 Macro Definition Documentation

5.1.2.1 __INDENT

```
#define __INDENT(
                f,
                n ) fprintf(f, "%.*s", n, " ")
```

__INDENT is a macro that prints n space characters to the FILE* f.

Parameters

f	- the f to print the spaces to
n	- the number of spaces to print to the file

5.2 include/tdd.h File Reference

```
#include <signal.h>
#include <stdarg.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>
```

Data Structures

- struct test_t
- struct testfn_s
- struct suite_t
- struct tdd_result_t
- struct suite_stats_t

Typedefs

- typedef struct test_t test_t
- typedef struct testfn_s testfn_t
- typedef struct suite_t suite_t
- typedef struct tdd_result_t tdd_result_t
- typedef struct suite_stats_t suite_stats_t

Functions

- void tdd_sigsegv_handler (int sig)
- void test_fail (test_t *t, char *msg)
- void test_error (test_t *t, char *msg)
- void test_start (test_t *t)
- void test_done (test_t *t)
- testfn_t * tdd_testfn_new (void *(*f)(void *t), char *name, char *desc)
- int tdd_testfn_del (testfn_t *tf)
- suite_t * suite_new ()
- void suite_reset (suite_t *s)
- int suite_del (suite_t *s)
- void suite_done (suite_t *s)

- void suite_add (suite_t *s, int n,...)
- int suite_addtest (suite_t *s, testfn_t *f)
- int suite_run (suite_t *s, bool fatal_failures)
- int suite_next (suite_t *s, bool fatal_failures)
- tdd_result_t * **tdd_result_new** (char *name, bool ok)
- int **tdd_result_del** (tdd_result_t *result)
- suite_stats_t * suite_stats (suite_t *s)
- int suite_stats_del (suite_stats_t *stats)

Variables

- volatile sig_atomic_t tdd_sigsegv_caught

5.2.1 Detailed Description

Author

Keefer Rourke mail@krourke.org

Date

08 Apr 2018

5.2.2 Typedef Documentation

5.2.2.1 suite_stats_t

```
typedef struct suite_stats_t suite_stats_t
```

Stats structure detailing results of test suite.

5.2.2.2 suite_t

```
typedef struct suite_t suite_t
```

Testing suite. Contains all tests, current runtime state, and the results of each test. May be used to contruct a suite_stats_t after running.

5.2.2.3 tdd_result_t

```
typedef struct tdd_result_t tdd_result_t
```

Test result. This structure holds the name of a test which ran, and indicates if the test passed.

5.2.2.4 test_t

```
typedef struct test_t test_t
```

Testing structure. This structure is the only parameter in all testing functions. If at any point during a testing function, unexpected bevahiour occurs or the test downright fails, you should call `test_error(t)` and `test_fail(t)` respectively.

5.2.2.5 testfn_t

```
typedef struct testfn_s testfn_t
```

Testing function.

5.2.3 Function Documentation

5.2.3.1 suite_add()

```
void suite_add (
            suite_t * s,
            int n,
             ...  )
```

`suite_add()` adds n testfn_t to the suite.

5.2.3.2 suite_addtest()

```
int suite_addtest (
            suite_t * s,
            testfn_t * f )
```

`suite_addtest()` adds a single `testfn_t` to the suite.

5.2.3.3 suite_del()

```
int suite_del (
            suite_t * s )
```

`suite_del()` frees memory allocated to a test suite.

Parameters

s	- a pointer to a `suite_t` test suite that is to be destroyed

Returns

5.2.3.4 suite_done()

```
void suite_done (
            suite_t * s )
```

`suite_done()` marks all the suite as having finished all tests.

5.2.3.5 suite_new()

```
suite_t* suite_new ( )
```

`suite_new()` creates and returns a new test suite.

Returns

A pointer to a fully intialized suite_t structure.

5.2.3.6 suite_next()

```
int suite_next (
            suite_t * s,
            bool fatal_failures )
```

`suite_next()` runs the next test in the suite.

Parameters

s	- the test suite to run
fatal_failures	- true indicates that the suite should abort testing if any test was marked as a failure

Returns

`EXIT_SUCCESS`, or, if `fatal_failures` is true, `EXIT_FAILURE` after the first failed test.

5.2.3.7 suite_reset()

```
void suite_reset (
            suite_t * s )
```

`suite_reset()` resets a `suite_t` to its initial state, as if it were never run.

Parameters

s	- a pointer to a `suite_t` test suite to be reset

5.2.3.8 suite_run()

```
int suite_run (
            suite_t * s,
            bool fatal_failures )
```

`suite_run()` runs all tests in the test array.

Parameters

s	- the test suite to run
fatal_failures	- true indicates that the suite should abort testing if any test was marked as a failure

Returns

`EXIT_SUCCESS`, or, if `fatal_failures` is true, `EXIT_FAILURE` after the first failed test.

5.2.3.9 suite_stats()

```
suite_stats_t* suite_stats (
            suite_t * s )
```

`suite_stats()` returns a `stats_t*` detailing the results of the testing.

5.2.3.10 suite_stats_del()

```
int suite_stats_del (
            suite_stats_t * stats )
```

`suite_stats_del()` frees memory allocated to a `stats_t*` returned by `suite_stats()`

5.2.3.11 tdd_sigsegv_handler()

```
void tdd_sigsegv_handler (
            int sig )
```

Crash handler.

5.2.3.12 tdd_testfn_del()

```
int tdd_testfn_del (
            testfn_t * tf )
```

`testfn_del()` frees memory allocated to a test function.

Parameters

tf	- a pointer to a `testfn_t` that is to be destroyed

Returns

`EXIT_SUCCESS`, otherwise `EXIT_FAILURE` if tf is `NULL`.

5.2.3.13 tdd_testfn_new()

```
testfn_t* tdd_testfn_new (
            void *(*)(void *t) f,
            char * name,
            char * desc )
```

`tdd_testfn_new()` creates and returns a pointer to an initialized testfn_t.

Parameters

f	- a pointer to a function that records information about a test within its single `test_t*` argument
name	- a character string identifier for the test; usually the name of the test function itself
desc	- a human readable description of the test; can be `NULL`

Returns

A pointer to a fully initialized testfn_t structure.

5.2.3.14 test_done()

```
void test_done (
            test_t * t )
```

`test_done()` marks the time at which the test finished. This may be useful for benchmarking and should be called before any test teardown code.

Parameters

t	- pointer to a `test_t` structure to capture the test finish time

5.2.3.15 test_error()

```
void test_error (
            test_t * t,
            char * msg )
```

`test_error()` marks the test as having encountered an error. Errors are identified as non-critical flaws in program function execution which do not prevent continuation of testing. Use `test_error()` to record unexpected but valid return values and similar flaws. To be called within a `testfn_t::fn`. Alternatively may be called through the `test_t::error` interface.

Parameters

t	- pointer to a `test_t` structure to capture the context of a test error
msg	- character string indicating the reason for error

5.2.3.16 test_fail()

```
void test_fail (
            test_t * t,
            char * msg )
```

`test_fail()` marks the test as failed with a message. Failures are identified as critical errors that will not allow testing to continue. Use `test_fail()` to catch fundamental errors in program function execution. To be called within a `testfn_t::fn`. Alternatively may be called through the `test_t::fail` interface.

Parameters

t	- pointer to a `test_t` structure to capture the context of a test failure
msg	- character string indicating the reason for failure

5.2.3.17 test_start()

```
void test_start (
            test_t * t )
```

`test_start()` marks the time at which the test started. This may be useful for benchmarking and should be called after any test setup code.

Parameters

t	- pointer to a `test_t` structure to capture the test finish time

5.2.4 Variable Documentation

5.2.4.1 tdd_sigsegv_caught

```
volatile sig_atomic_t tdd_sigsegv_caught
```

`tdd_sigsegv_caught` is a counter for the number of crashes that were encountered in the test suite.

5.3 include/timeutil.h File Reference

Time format and manipulation functions and macros for libtdd.

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

Functions

- struct timespec **__timespec_minus** (struct timespec *a, struct timespec *b)

5.3.1 Detailed Description

Time format and manipulation functions and macros for libtdd.

Author

Keefer Rourke mail@krourke.org

Date

08 Apr 2018

Index