Institute for System Programming of Russian Academy of Sciences

VERIFICATION CENTER OF THE OPERATING SYSTEM Linux

OLVER Reports: Reference Guide

Version	Date	Description of the changes	Author
1.0	28-Dec-2006	The first version.	Alexey Khoroshilov

Introduction

This document describes test result reports generated by the OLVER test suite[1]. The goal of the test suite is to check the requirements of the *Linux Standard Base Core Specification 3.1* [2] to the functions of the application binary interface contained in the sections *III Base Libraries* and *IV Utility Libraries*. The sections mentioned above define the requirements to the presence and the functionality of 1532 functions of the Linux application binary interface.

The functions having similar functionality are collected into *subsystems*. Distribution of the functions among subsystems is described at the Linux Verification Center web site [1].

There are the following format conventions in this document. Names of files and identifiers are shown in `fixed-width font`. Parts of file path that depends on some conditions are marked by angle brackets (for example, `<scenario_name>.utt`).

Test suite execution

Execution of the OLVER test suite consists in sequential running of test scenarios enumerating in the `/opt/olver/testplan` file. Each test scenario is typically intended for testing a group of functions associated by common functionality.

Details concerning a run of a test scenario are saved to a file named *test trace*. The file name is `/var/opt/olver/<TIMESTAMP>/<scenario_name>.utt`, where `<TIMESTAMP>` is the time of test scenario launching in the form of "YEAR-MONTH-DAY_HOUR-MINUTE-SECOND". A set of HTML reports is generated automatically from the test traces. There are the following reports available:

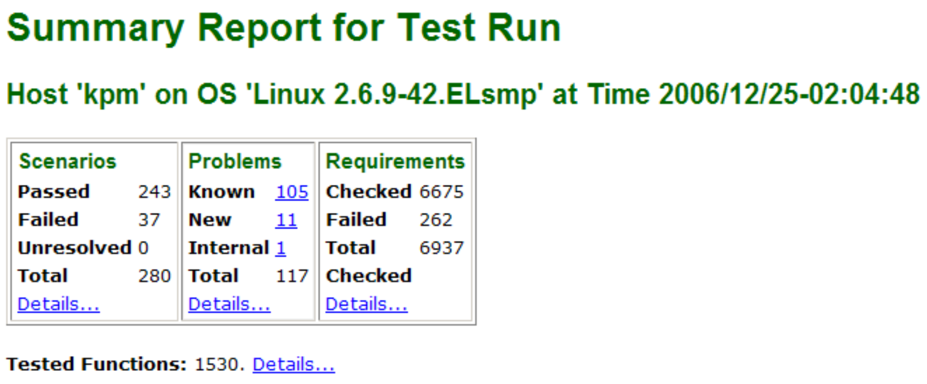
- Summary test report
- Detailed test report
  - Failures report
  - Test scenarios report
  - Branches coverage report
- Requirements coverage report

Contents of these reports are described below.

Summary test report

Summary test report is generated at the `/var/opt/olver/<TIMESTAMP>/summary.htm` file. The report consists of three parts: a header, a list of the problems detected, and a list of the functions tested.

The header contains statictics of the test execution (Picture 1).



Summary Report for Test Run

Host 'kpm' on OS 'Linux 2.6.9-42.ELsmp' at Time 2006/12/25-02:04:48

Scenarios		Problems		Requirements	
Passed	243	**Known**	105	**Checked**	6675
Failed	37	**New**	11	**Failed**	262
Unresolved	0	**Internal**	1	**Total Checked**	6937
Total	280	**Total**	117		
Details...		Details...		Details...	

Tested Functions: 1530. Details...

Picture 1: Header of the summary test report

The first line of the header describes a host name of the system under test, an identifier of operating system, and launching time of the test execution. The next table contains quantitative properties of the test execution.

The first column ('Scenarios') includes the following information:

- Passed – a number of tests scenarios that does not detect a failure;

- Failed – a number of tests scenarios that does detect a failure;
- Unresolved – a number of tests scenarios with unknown status of the execution;
- Total – a total number of test scenarios executed.

The 'Details...' hyperlink of the scenarios column leads to the detailed test report discussed in the next section.

The second column ('Problems') describes the detected problems:

- Known – a number of the problems identified as an instance of a know bug;
- New – a number of the detected inconsistencies between the specification and the system under test, which are not identified as an instance of a know bug;
- Internal – a number of the detected problems, which are a consequence of the bug in the test suite rather than a consequence of the bug in the system under test;
- Total – a total number of the detected problems.

The 'Details...' hyperlink of the problems column leads to the problems section of the summary test report.

The last column ('Requirements') contains test execution statistics concerning coverage of elementary requirements of the LSB Core 3.1 specification to the system under test. It includes the following lines:

- Checked – a number of the elementary requirements successfully checked during the test execution;
- Failed – a number of the elementary requirements, violation of which was detected during the test execution;
- Total checked - a total number of the elementary requirements checked during the test execution.

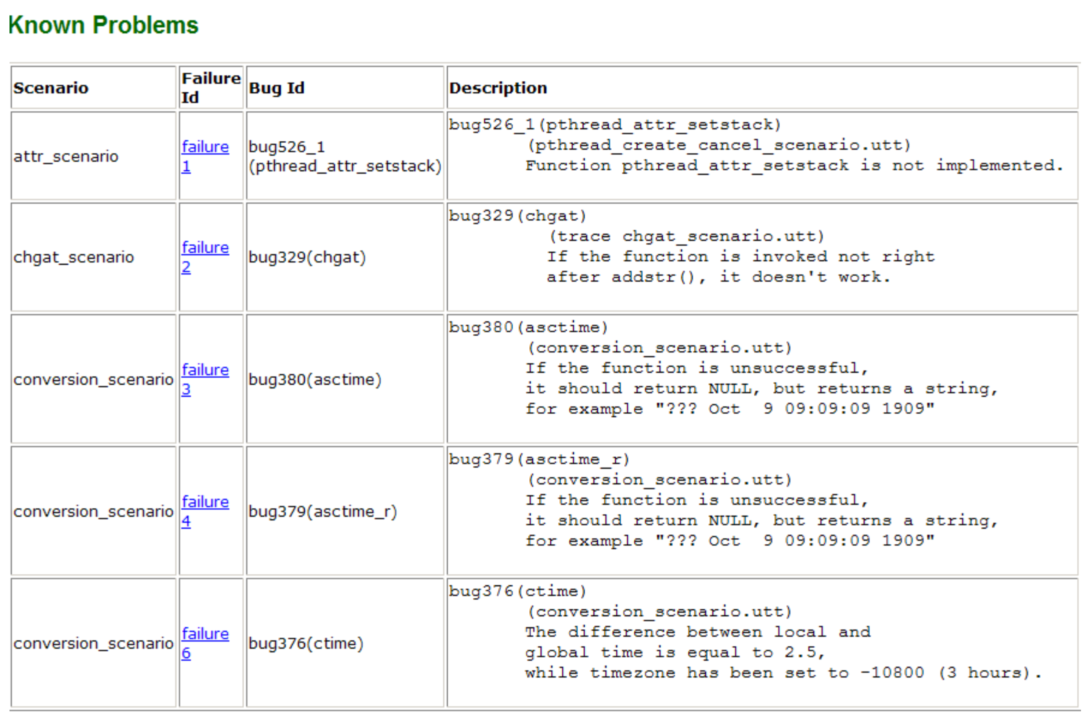
The 'Details...' hyperlink of the requirements column leads to the requirements coverage report.

The last line of the header contains a number of functions under test, which ware tested during the test execution. The 'Details...' hyperlink at this line leads to the list of the tested functions located in the corresponding section of the summary test report.

The problems section follows the header of the summary test report. It contains up to three tables describing the detected problems of each kind (Known, New and Internal).

The 'Known Problems' table (Picture 2) has four coloumns:

- Scenario – a name of the test scenario detected the given problem;



Known Problems

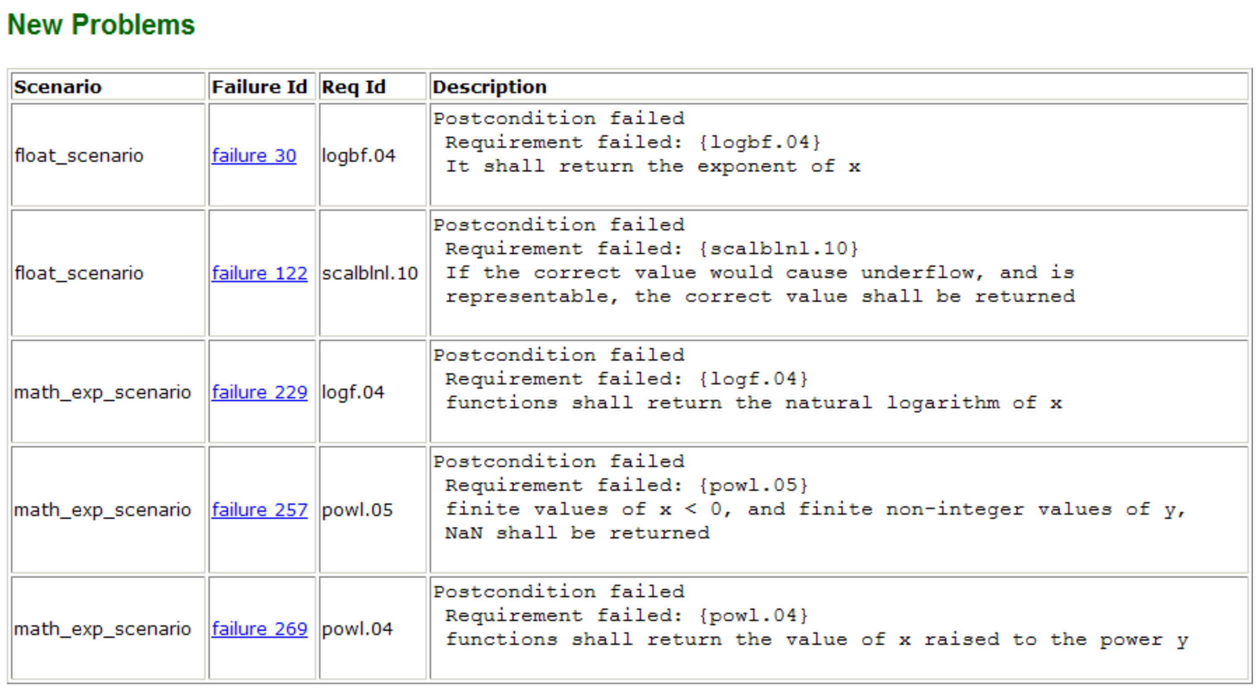
Scenario	Failure Id	Bug Id	Description
attr_scenario	failure 1	bug526_1 (pthread_attr_setstack)	`bug526_1(pthread_attr_setstack) (pthread_create_cancel_scenario.utt) Function pthread_attr_setstack is not implemented.`
chgat_scenario	failure 2	bug329(chgat)	`bug329(chgat) (trace chgat_scenario.utt) If the function is invoked not right after addstr(), it doesn't work.`
conversion_scenario	failure 3	bug380(asctime)	`bug380(asctime) (conversion_scenario.utt) If the function is unsuccessful, it should return NULL, but returns a string, for example "??? Oct  9 09:09:09 1909"`
conversion_scenario	failure 4	bug379(asctime_r)	`bug379(asctime_r) (conversion_scenario.utt) If the function is unsuccessful, it should return NULL, but returns a string, for example "??? Oct  9 09:09:09 1909"`
conversion_scenario	failure 6	bug376(ctime)	`bug376(ctime) (conversion_scenario.utt) The difference between local and global time is equal to 2.5, while timezone has been set to -10800 (3 hours).`

Picture 2: Known problems table of the summary test report

- Failure Id – an identifier of the problem in the detailed test report and a hyperlink to the detailed description of the problem in that report;
- Bug Id – an identifier of the known bug, corresponding to the given problem;
- Description – a brief description of the given problem.

The 'New Problems' table (Picture 3) contains information about detected inconsistencies between the specification and the system under test, which are not identified as an instance of a know bug. It includes 4 coloumns:

- Scenario – a name of the test scenario detected the given problem;
- Failure Id – an identifier of the problem in the detailed test report and a hyperlink to the detailed description of the problem in that report;
- Req Id – an identifier of the elementary requirement, violation of which is the reason of the given problem;
- Description – a brief description of the violated requirement.



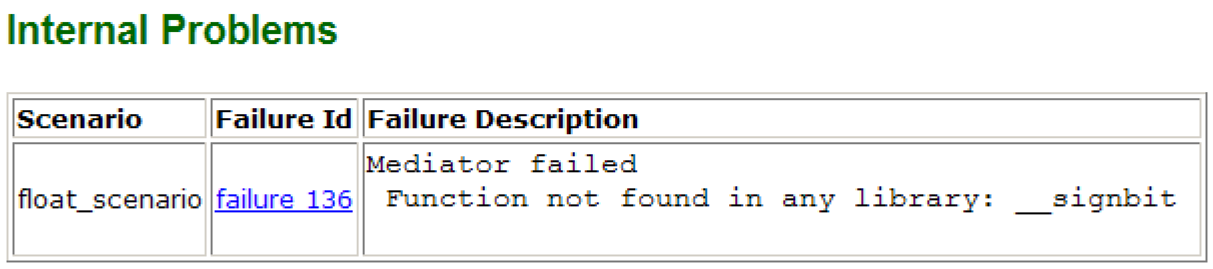
New Problems

Scenario	Failure Id	Req Id	Description
float_scenario	failure 30	logbf.04	Postcondition failed Requirement failed: {logbf.04} It shall return the exponent of x
float_scenario	failure 122	scalblnl.10	Postcondition failed Requirement failed: {scalblnl.10} If the correct value would cause underflow, and is representable, the correct value shall be returned
math_exp_scenario	failure 229	logf.04	Postcondition failed Requirement failed: {logf.04} functions shall return the natural logarithm of x
math_exp_scenario	failure 257	powl.05	Postcondition failed Requirement failed: {powl.05} finite values of x < 0, and finite non-integer values of y, NaN shall be returned
math_exp_scenario	failure 269	powl.04	Postcondition failed Requirement failed: {powl.04} functions shall return the value of x raised to the power y

Picture 3: New problems table of the summary test report

The 'Internal Problems' table (Picture 4) describes the detected problems, which are a consequence of the bug in the test suite rather than a consequence of the bug in the system under test. The table include three coloumns:

- Scenario – a name of the test scenario detected the given problem;
- Failure Id – an identifier of the problem in the detailed test report and a hyperlink to the detailed description of the problem in that report;
- Req Id – an identifier of the elementary requirement, violation of which is the reason of the given problem;
- Failure Description – a brief description of the problem.

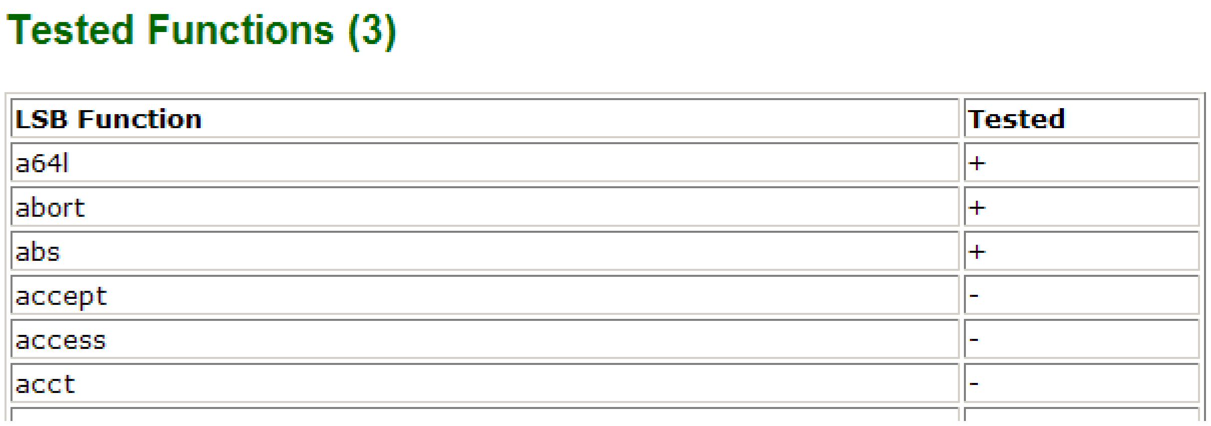


Internal Problems

Scenario	Failure Id	Failure Description
float_scenario	failure 136	Mediator failed Function not found in any library: __signbit

Picture 4: Internal problems table of the summary test report

The summary test report ends with the tested functions section (Picture 5), which contains a list of interface functions from the LSB Core Generic 3.1 specification. If a function was tested during the test execution it is marked by the plus sign, otherwise it is marked by the minus sign.



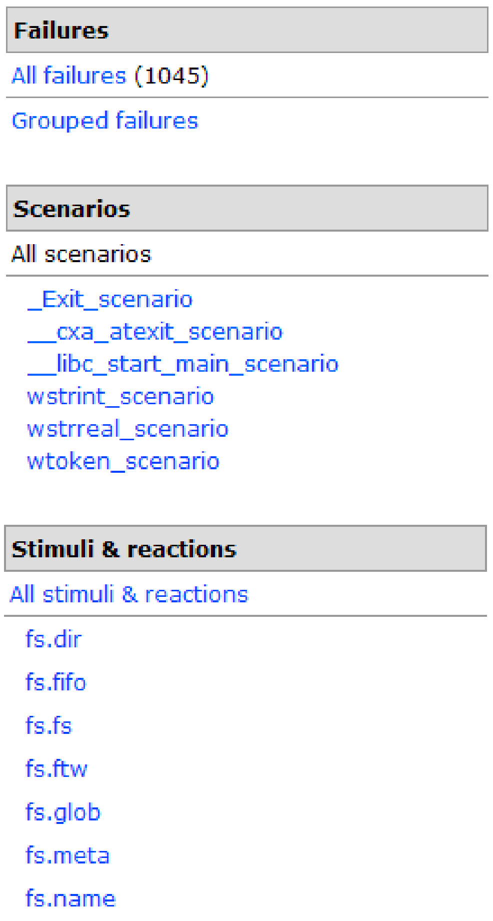
Tested Functions (3)

LSB Function	Tested
a64l	+
abort	+
abs	+
accept	-
access	-
acct	-

Picture 5: Tested functions section of the summary test report

Detailed test report

Detailed test report is available by the 'Details...' hyperlink of the scenarios coloumn of the summary test report header. This hyperlink leads to the `/var/opt/olver/<TIMESTAMP>/report/index.html` file, which is an entry page of the report. Other pages are available by hyperlinks from the navigation bar placed at the lest side of the report. The navigation bar consists of three sections: ‘Failures’, ‘Scenarios’, and ‘Stimuli & reactions’ (Picture 6).



Picture 6: Navigation bar of the detailed test report

The 'Failures' section contains hyperlinks to different representations of detected problems. The 'All failures' link leads to a plain table of the problems and the 'Grouped failures' link leads to a tree of the problems grouped by the rules that are described below.

The 'Scenarios' section includes a link to a summary report of test scenarios execution status and links to detailed reports of each scenario executed.

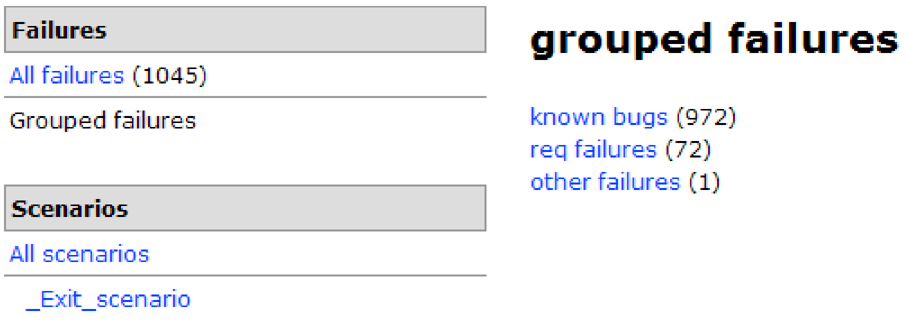
The 'Stimuli & reactions' section allows to analyse reports concerning functionality branches coverage. The 'All stimuli & reactions' link leads to a summary coverage report. The links under subsystem names lead to reports about coverage of the corresponding subsystem.

Failures report

A table available on the 'All failures' link contains a short description of each detected problem and a hyperlink to the detailed report concerning the corresponding problem.

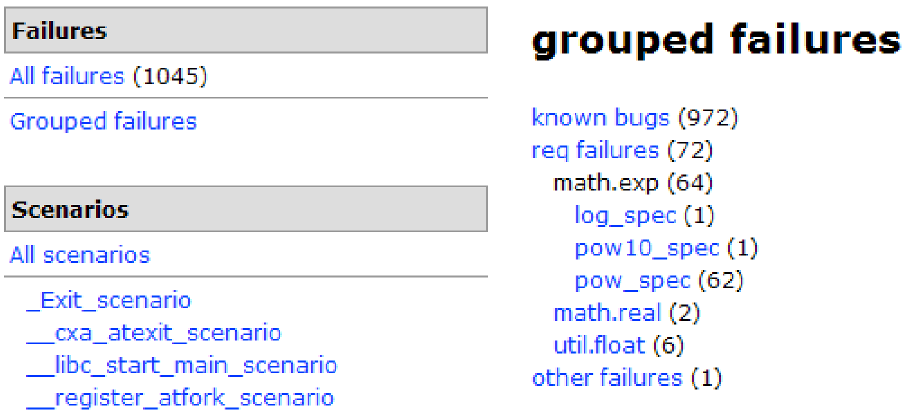
The 'Grouped failures' representation groups the detected problems accroding the following rules. Firstly all problems are distributed by the three groups (Picture 7):

- known bugs – problems identified as an instance of a known bug;
- req failures – problems corresponding to an inconsistency between the specification and the system under test, which is not identified as an instance of a know bug;
- other failures - problems, which are a consequence of the bug in the test suite rather than a consequence of the bug in the system under test.



Picture 7: The first level of the grouped failures tree

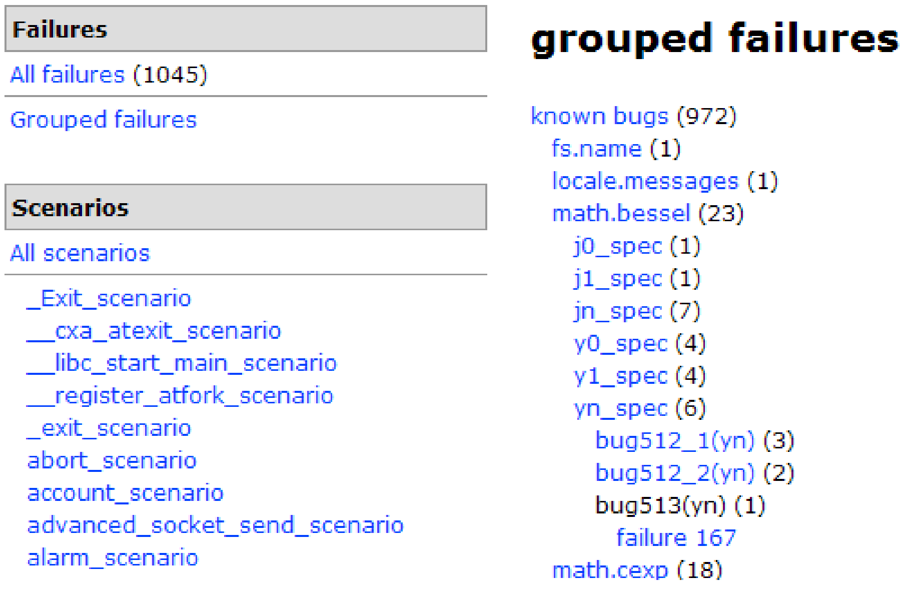
At the next two levels problems are grouped according subsystems and functions, where the problem occurs (Picture 8). These levels are common for all the groups of the first level.



Picture 8: The second and third levels of the grouped failures tree

The fourth level is available for the 'known bugs' category only. At this level all problems identified as an instance of the same bug are united (Picture 9).

Leaf nodes in the grouped failures tree contain a hyperlink to a detailed report of the corresponding problem. This report provides a brief description of the problem and information about the failed test case including input and output parameter values. If the failure is identified



Failures

All failures (1045)

Grouped failures

Scenarios

All scenarios

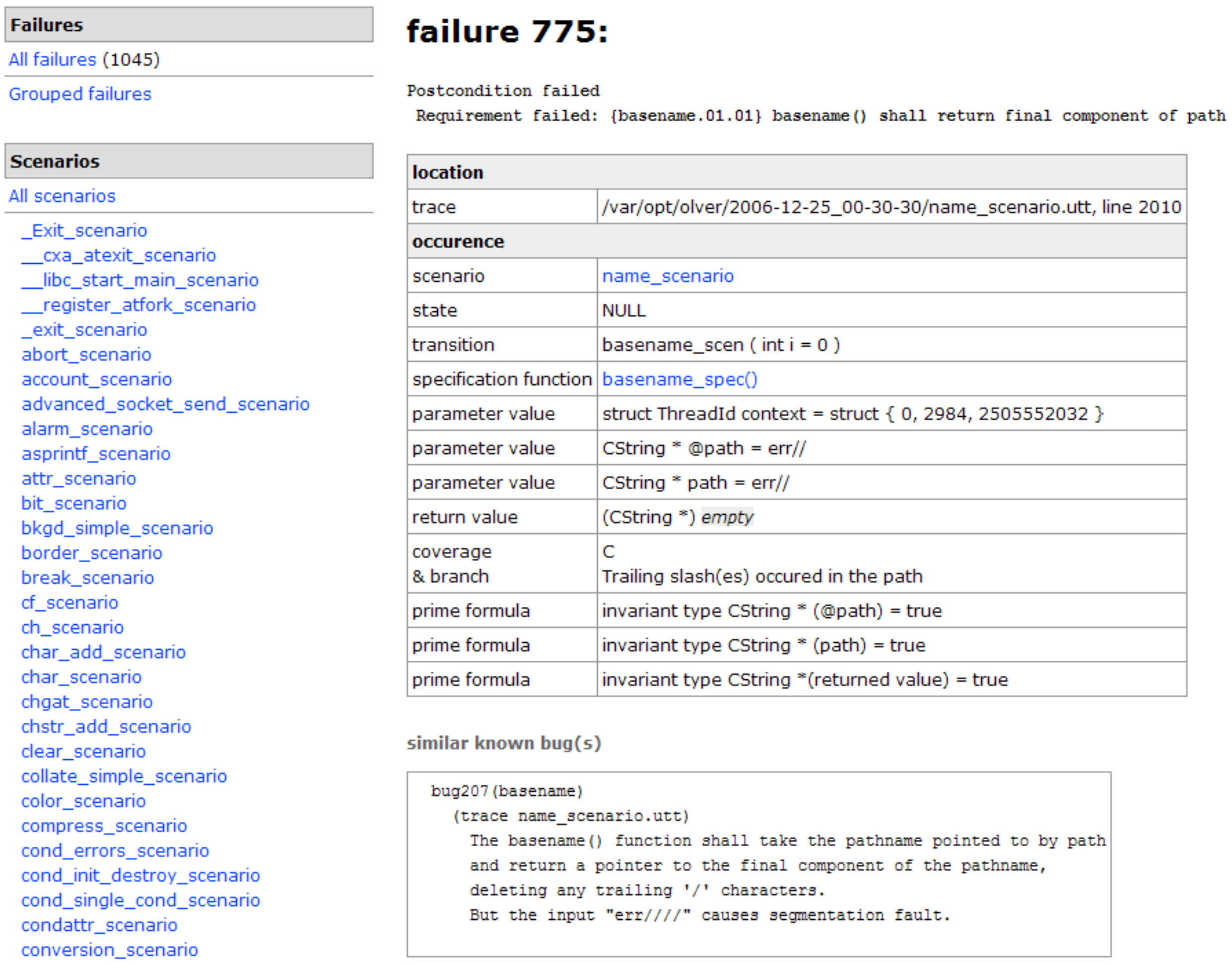
- _Exit_scenario
- __cxa_atexit_scenario
- __libc_start_main_scenario
- __register_atfork_scenario
- _exit_scenario
- abort_scenario
- account_scenario
- advanced_socket_send_scenario
- alarm_scenario

grouped failures

- known bugs (972)
  - fs.name (1)
  - locale.messages (1)
  - math.bessel (23)
    - j0_spec (1)
    - j1_spec (1)
    - jn_spec (7)
    - y0_spec (4)
    - y1_spec (4)
    - yn_spec (6)
      - bug512_1(yn) (3)
      - bug512_2(yn) (2)
      - bug513(yn) (1)
        - failure 167
  - math.cexp (18)

Picture 9: The fourth level of the 'known bugs' category

as an instance of a known bug a description of the bug is also placed at the end of the detailed failure report (Picture 10).



Failures

All failures (1045)

Grouped failures

Scenarios

All scenarios

- _Exit_scenario
- __cxa_atexit_scenario
- __libc_start_main_scenario
- __register_atfork_scenario
- _exit_scenario
- abort_scenario
- account_scenario
- advanced_socket_send_scenario
- alarm_scenario
- asprintf_scenario
- attr_scenario
- bit_scenario
- bkgd_simple_scenario
- border_scenario
- break_scenario
- cf_scenario
- ch_scenario
- char_add_scenario
- char_scenario
- chgat_scenario
- chstr_add_scenario
- clear_scenario
- collate_simple_scenario
- color_scenario
- compress_scenario
- cond_errors_scenario
- cond_init_destroy_scenario
- cond_single_cond_scenario
- condattr_scenario
- conversion_scenario
- created_timers_scenario

failure 775:

```
Postcondition failed
 Requirement failed: {basename.01.01} basename() shall return final component of path
```

location	
trace	/var/opt/olver/2006-12-25_00-30-30/name_scenario.utt, line 2010
occurence	
scenario	name_scenario
state	NULL
transition	basename_scen (int i = 0)
specification function	basename_spec()
parameter value	struct ThreadId context = struct { 0, 2984, 2505552032 }
parameter value	CString * @path = err//
parameter value	CString * path = err//
return value	(CString *) *empty*
coverage & branch	C Trailing slash(es) occured in the path
prime formula	invariant type CString * (@path) = true
prime formula	invariant type CString * (path) = true
prime formula	invariant type CString *(returned value) = true

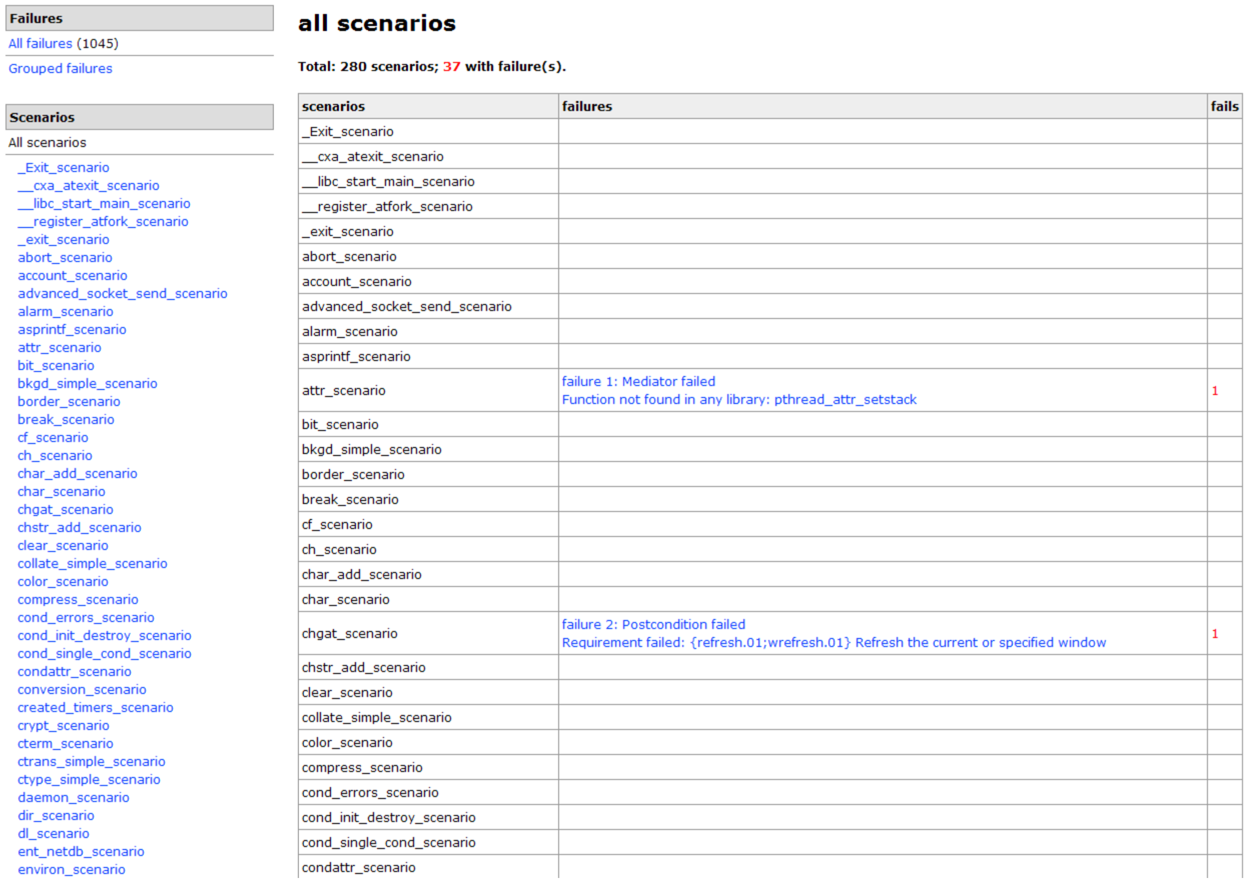
similar known bug(s)

```
bug207(basename)
  (trace name_scenario.utt)
    The basename() function shall take the pathname pointed to by path
    and return a pointer to the final component of the pathname,
    deleting any trailing '/' characters.
    But the input "err////" causes segmentation fault.
```

Picture 10: Detailed failure report

Test scenarios report

Summary test scenarios report is available by the 'All scenarios' hyperlink at the navigation bar. The first line (Picture 11) demonstrates a number of executed test scenarios and a number of failed test scenarios. The second element of the report is a table, which contains a total number of the problems detected by the given scenario and a hyperlink to the first problem.



Failures
All failures (1045)
Grouped failures

Scenarios
All scenarios
_Exit_scenario
__cxa_atexit_scenario
__libc_start_main_scenario
__register_atfork_scenario
_exit_scenario
abort_scenario
account_scenario
advanced_socket_send_scenario
alarm_scenario
asprintf_scenario
attr_scenario
bit_scenario
bkgd_simple_scenario
border_scenario
break_scenario
cf_scenario
ch_scenario
char_add_scenario
char_scenario
chgat_scenario
chstr_add_scenario
clear_scenario
collate_simple_scenario
color_scenario
compress_scenario
cond_errors_scenario
cond_init_destroy_scenario
cond_single_cond_scenario
condattr_scenario
conversion_scenario
created_timers_scenario
crypt_scenario
cterm_scenario
ctrans_simple_scenario
ctype_simple_scenario
daemon_scenario
dir_scenario
dl_scenario
ent_netdb_scenario
environ_scenario

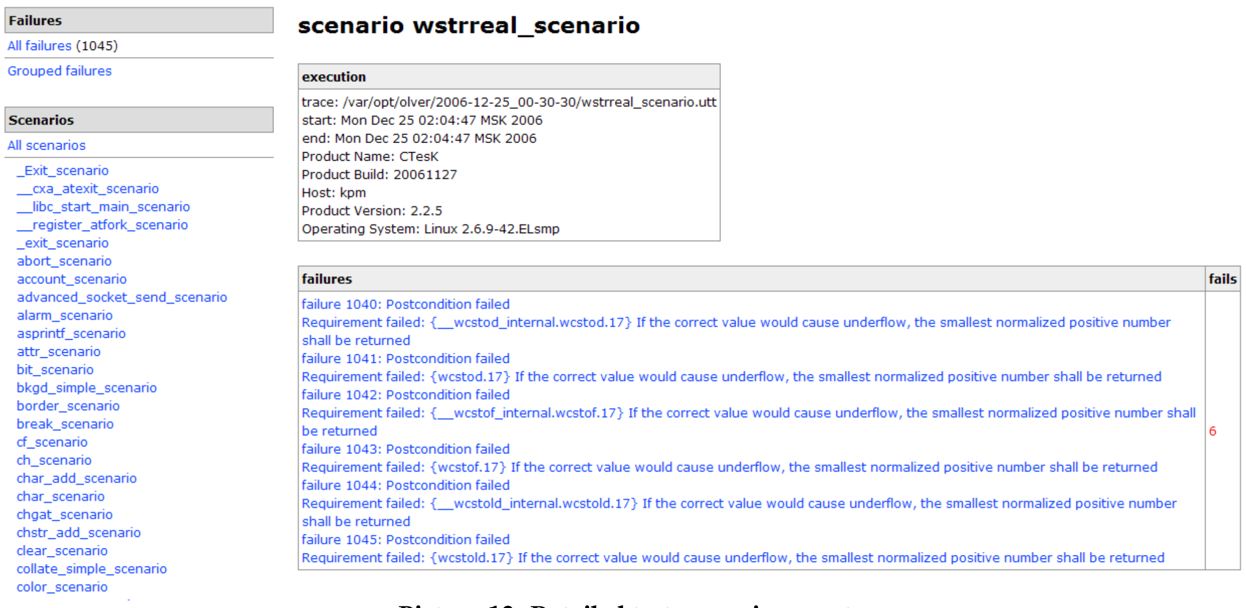
all scenarios

Total: 280 scenarios; 37 with failure(s).

scenarios	failures	fails
_Exit_scenario		
__cxa_atexit_scenario		
__libc_start_main_scenario		
__register_atfork_scenario		
_exit_scenario		
abort_scenario		
account_scenario		
advanced_socket_send_scenario		
alarm_scenario		
asprintf_scenario		
attr_scenario	failure 1: Mediator failed Function not found in any library: pthread_attr_setstack	1
bit_scenario		
bkgd_simple_scenario		
border_scenario		
break_scenario		
cf_scenario		
ch_scenario		
char_add_scenario		
char_scenario		
chgat_scenario	failure 2: Postcondition failed Requirement failed: {refresh.01;wrefresh.01} Refresh the current or specified window	1
chstr_add_scenario		
clear_scenario		
collate_simple_scenario		
color_scenario		
compress_scenario		
cond_errors_scenario		
cond_init_destroy_scenario		
cond_single_cond_scenario		
condattr_scenario		

Picture 11: Summary test scenario report

Other hyperlinks at the 'Scenario' section of the navigation bar lead to detailed reports concerning the corresponding test scenario (Picture 12). These reports provide information about



Failures
All failures (1045)
Grouped failures

Scenarios
All scenarios
_Exit_scenario
__cxa_atexit_scenario
__libc_start_main_scenario
__register_atfork_scenario
_exit_scenario
abort_scenario
account_scenario
advanced_socket_send_scenario
alarm_scenario
asprintf_scenario
attr_scenario
bit_scenario
bkgd_simple_scenario
border_scenario
break_scenario
cf_scenario
ch_scenario
char_add_scenario
char_scenario
chgat_scenario
chstr_add_scenario
clear_scenario
collate_simple_scenario
color_scenario

scenario wstrreal_scenario

execution
trace: /var/opt/olver/2006-12-25_00-30-30/wstrreal_scenario.utt start: Mon Dec 25 02:04:47 MSK 2006 end: Mon Dec 25 02:04:47 MSK 2006 Product Name: CTesK Product Build: 20061127 Host: kpm Product Version: 2.2.5 Operating System: Linux 2.6.9-42.ELsmp

failures	fails
failure 1040: Postcondition failed Requirement failed: {__wcstod_internal.wcstod.17} If the correct value would cause underflow, the smallest normalized positive number shall be returned failure 1041: Postcondition failed Requirement failed: {wcstod.17} If the correct value would cause underflow, the smallest normalized positive number shall be returned failure 1042: Postcondition failed Requirement failed: {__wcstof_internal.wcstof.17} If the correct value would cause underflow, the smallest normalized positive number shall be returned failure 1043: Postcondition failed Requirement failed: {wcstof.17} If the correct value would cause underflow, the smallest normalized positive number shall be returned failure 1044: Postcondition failed Requirement failed: {__wcstold_internal.wcstold.17} If the correct value would cause underflow, the smallest normalized positive number shall be returned failure 1045: Postcondition failed Requirement failed: {wcstold.17} If the correct value would cause underflow, the smallest normalized positive number shall be returned	6

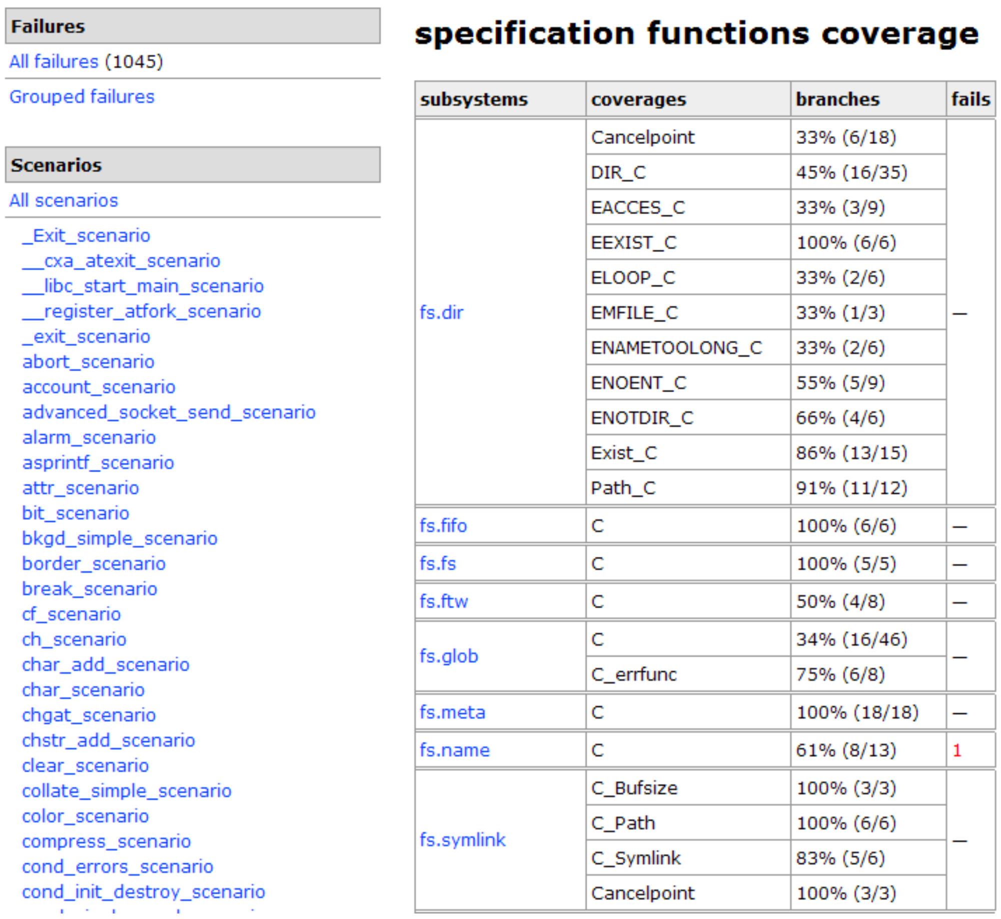
Picture 12: Detailed test scenario report

execution time and environment of the given scenario. If the scenario detects any problem the report enumerates all these problems.

Branches coverage report

The UniTesK [3] testing technology, which is a basis of the OLVER test suite, supports the testing quality measurement technique based on branches of functionality. According this technique a test developer defines branches of functionality for each function under test and coverage of these branches is measured during test execution. More than one set of branches of functionality can be defined for a function. A set of branches is called a *coverage*. More details on this testing quality measurement technique is presented in the document «Formal Specification-Based Testing: Getting Started» [4] and at the web-site [3].

The 'All stimuli & reactions' link leads to a summary branches coverage report (Picture 13). The report provides a table describing quality of testing in terms of branches of functionality. For each subsystem enumerated in the first column there is a list of the defined coverages in the second column. Numbers in brackets from the third column mean a number of covered and a total number of branches defined in the given subsystem and in the given coverage. A percentage illustrates a proportion of the numbers. The last optional column contains a number of problems detected in the corresponding subsystem.



Failures

All failures (1045)

Grouped failures

Scenarios

All scenarios

_Exit_scenario
__cxa_atexit_scenario
__libc_start_main_scenario
__register_atfork_scenario
_exit_scenario
abort_scenario
account_scenario
advanced_socket_send_scenario
alarm_scenario
asprintf_scenario
attr_scenario
bit_scenario
bkgd_simple_scenario
border_scenario
break_scenario
cf_scenario
ch_scenario
char_add_scenario
char_scenario
chgat_scenario
chstr_add_scenario
clear_scenario
collate_simple_scenario
color_scenario
compress_scenario
cond_errors_scenario
cond_init_destroy_scenario

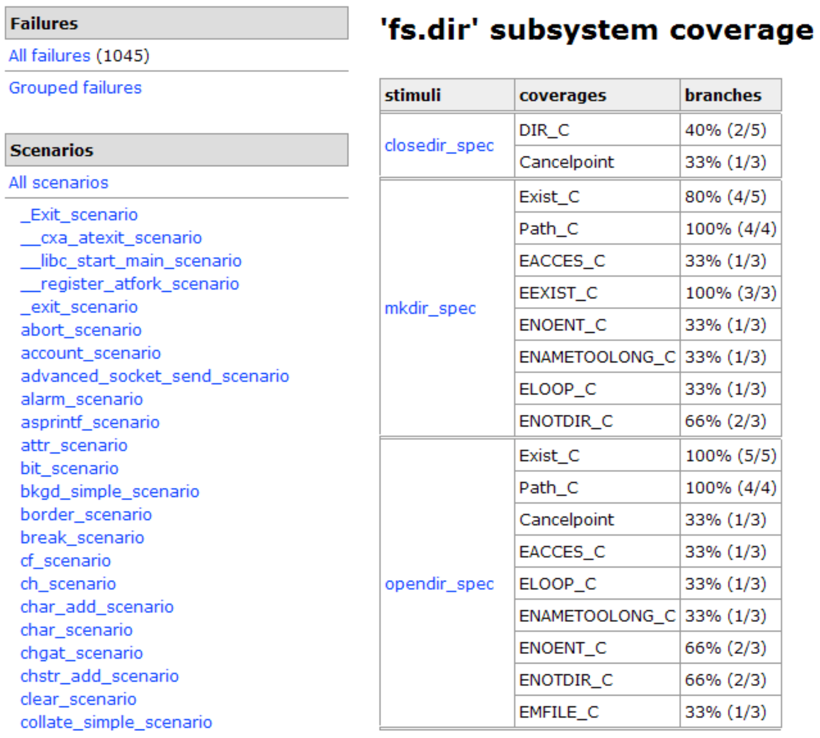
specification functions coverage

subsystems	coverages	branches	fails
fs.dir	Cancelpoint	33% (6/18)	—
	DIR_C	45% (16/35)	
	EACCES_C	33% (3/9)	
	EEXIST_C	100% (6/6)	
	ELOOP_C	33% (2/6)	
	EMFILE_C	33% (1/3)	
	ENAMETOOLONG_C	33% (2/6)	
	ENOENT_C	55% (5/9)	
	ENOTDIR_C	66% (4/6)	
	Exist_C	86% (13/15)	
	Path_C	91% (11/12)	
fs.fifo	C	100% (6/6)	—
fs.fs	C	100% (5/5)	—
fs.ftw	C	50% (4/8)	—
fs.glob	C	34% (16/46)	—
	C_errfunc	75% (6/8)	
fs.meta	C	100% (18/18)	—
fs.name	C	61% (8/13)	1
fs.symlink	C_Bufsize	100% (3/3)	—
	C_Path	100% (6/6)	
	C_Symlink	83% (5/6)	
	Cancelpoint	100% (3/3)	

Picture 13: Summary branches coverage report

The hyperlinks under subsystem names lead to branches coverage report concerning the given subsystem (Picture 14). It has the same structure as the previous one except for substitution of subsystems with functions belonging the given subsystem.

The most detailed branches coverage report is a report concerning a concrete function (Picture 15). It is available by the hyperlink under a name of the corresponding function in the first column of subsystem branches coverage report. The report describes all coverages defined



Failures

All failures (1045)

Grouped failures

Scenarios

All scenarios

- _Exit_scenario
- __cxa_atexit_scenario
- __libc_start_main_scenario
- __register_atfork_scenario
- _exit_scenario
- abort_scenario
- account_scenario
- advanced_socket_send_scenario
- alarm_scenario
- asprintf_scenario
- attr_scenario
- bit_scenario
- bkgd_simple_scenario
- border_scenario
- break_scenario
- cf_scenario
- ch_scenario
- char_add_scenario
- char_scenario
- chgat_scenario
- chstr_add_scenario
- clear_scenario
- collate_simple_scenario

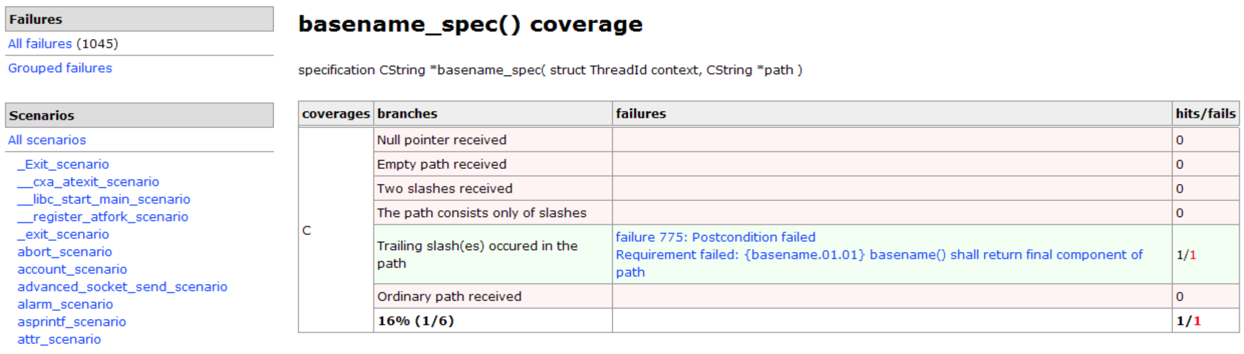
'fs.dir' subsystem coverage

stimuli	coverages	branches
closedir_spec	DIR_C	40% (2/5)
	Cancelpoint	33% (1/3)
mkdir_spec	Exist_C	80% (4/5)
	Path_C	100% (4/4)
	EACCES_C	33% (1/3)
	EEXIST_C	100% (3/3)
	ENOENT_C	33% (1/3)
	ENAMETOOLONG_C	33% (1/3)
	ELOOP_C	33% (1/3)
	ENOTDIR_C	66% (2/3)
opendir_spec	Exist_C	100% (5/5)
	Path_C	100% (4/4)
	Cancelpoint	33% (1/3)
	EACCES_C	33% (1/3)
	ELOOP_C	33% (1/3)
	ENAMETOOLONG_C	33% (1/3)
	ENOENT_C	66% (2/3)
	ENOTDIR_C	66% (2/3)
	EMFILE_C	33% (1/3)

Picture 14: Subsystem branches coverage report

in the given function. There are names of the coverages in the first column. For each coverage all the branches are enumerated in the second column. There is a coverage statistics at the bottom of the list. The raws corresponding to covered branches are marked by green background, the raws corresponding to uncovered branches are marked by red background.

The last column of the table contains a number of calls corresponding to the given branch of functionality. If there are problems detected in the function the additional 'failures' column describes a distribution of the problems between the branches.



Failures

All failures (1045)

Grouped failures

Scenarios

All scenarios

- _Exit_scenario
- __cxa_atexit_scenario
- __libc_start_main_scenario
- __register_atfork_scenario
- _exit_scenario
- abort_scenario
- account_scenario
- advanced_socket_send_scenario
- alarm_scenario
- asprintf_scenario
- attr_scenario

basename_spec() coverage

specification CString *basename_spec(struct ThreadId context, CString *path)

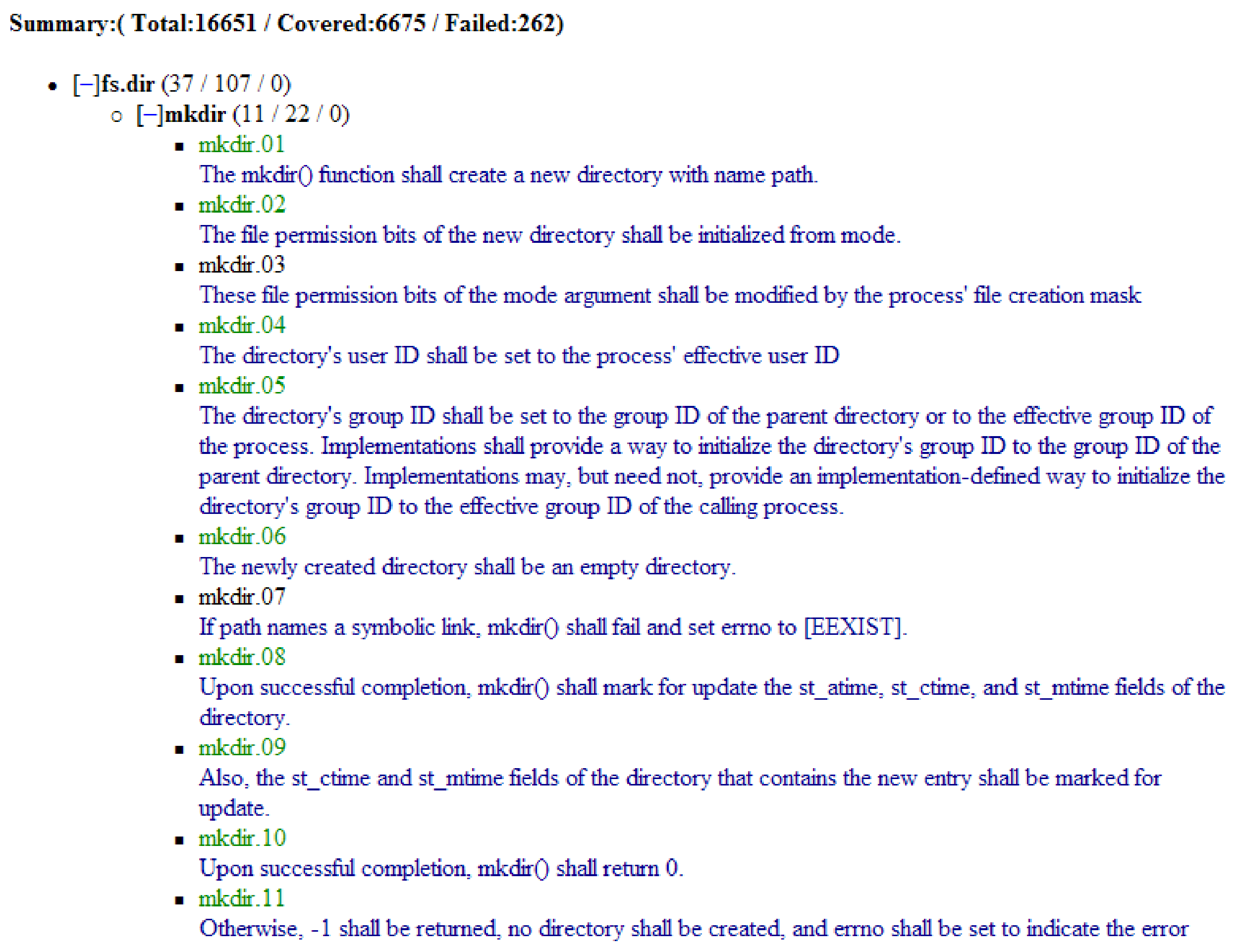
coverages	branches	failures	hits/fails
C	Null pointer received		0
	Empty path received		0
	Two slashes received		0
	The path consists only of slashes		0
	Trailing slash(es) occured in the path	failure 775: Postcondition failed Requirement failed: {basename.01.01} basename() shall return final component of path	1/1
	Ordinary path received		0
	16% (1/6)		**1/1**

Picture 15: Function branches coverage report

Requirements coverage report

Requirements coverage report is available by the 'Details...' hyperlink of the requirements coloumn of the summary test report header and it is placed at the `/var/opt/olver/<TIMESTAMP>/result.htm` file.

The first line of the report contains summary coverage statictics (Picture 16). The first number (Total) is a total number of elementary requirements of the LSB Core 3.1 specification to the system under test. The second number (Covered) is a numbed of tested requirements and the last one (Failed) is a number of the elementary requirements, violation of which was detected during the test execution.



Summary:(Total:16651 / Covered:6675 / Failed:262)

- [–]**fs.dir** (37 / 107 / 0)
  - [–]**mkdir** (11 / 22 / 0)
    - mkdir.01
      The mkdir() function shall create a new directory with name path.
    - mkdir.02
      The file permission bits of the new directory shall be initialized from mode.
    - mkdir.03
      These file permission bits of the mode argument shall be modified by the process' file creation mask
    - mkdir.04
      The directory's user ID shall be set to the process' effective user ID
    - mkdir.05
      The directory's group ID shall be set to the group ID of the parent directory or to the effective group ID of the process. Implementations shall provide a way to initialize the directory's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the directory's group ID to the effective group ID of the calling process.
    - mkdir.06
      The newly created directory shall be an empty directory.
    - mkdir.07
      If path names a symbolic link, mkdir() shall fail and set errno to [EEXIST].
    - mkdir.08
      Upon successful completion, mkdir() shall mark for update the st_atime, st_ctime, and st_mtime fields of the directory.
    - mkdir.09
      Also, the st_ctime and st_mtime fields of the directory that contains the new entry shall be marked for update.
    - mkdir.10
      Upon successful completion, mkdir() shall return 0.
    - mkdir.11
      Otherwise, -1 shall be returned, no directory shall be created, and errno shall be set to indicate the error

Picture 16: Requirements coverage report

The main part of the report is a tree of elementary requirements, where:

- nodes of the first level correspond to subsystems and contain requirements coverage statistics in the form (Total / Covered / Failed);
  - nodes of the second level correspond to functions and contain requirements coverage statistics in the form (Total / Covered / Failed);
    - nodes of the third level contain elementary requirements including an identifier and a text of the requirement.

The tree view allows to hide and unhide all the nodes placed under any nonleaf node. To do that someone shall click at the [+]/[–] sign near the node.

Identifiers of requirements, which were not checked during the test execution, have black colour. Identifiers of requirements, which were successfully checked during the test execution, have green colour. Identifiers of requirements, violation of which was detected during the test execution, have red colour and are marked by (FAILED) sign. Grey identifiers correspond to uncheckable requirements such as requirements to application using the system under test.

OLVER reports usage guide

Depending on goals of using the OLVER test suite we recommend to pay attention to different kinds of the OLVER reports. The given section provides several recomendations on usage of the OLVER reports.

If you use the test suite for **obtaining information about conformance** a system under test to the LSB Core 3.1 specification, first of all you should analyse a summary test report. The header of the report provides the following statistics:

- numbers of detected problems of different kinds (the 'Problems' column);
- a number of failed test scenarios (the 'Scenarios' column, the 'Failed' line);
- a number of the elementary requirements, violation of which was detected during the test execution (the 'Requirements' column, the 'Failed' line).

More detailed information about detected problems is available in the 'Problems' section of the same report. In addition, we recommend to pay attention to a distribution of the problems between subsystems and functions, which is described in the 'Grouped failures' section of the detailed test report.

If you use the OLVER test suite for **detailed analysis of inconsistencies** between a system under test and the LSB Core 3.1 specification, the most important report is a detailed test report and the 'Failures' section. This section describes both a distribution of the problems between subsystems and functions and details of each problems in particularly. In addition to static HTML reports we recommend to use the dynamic trace player tool available as a part of the CTesK toolkit [3].

If the **information about quality of testing** is important to you, the following two reports shall be investigated:

- requirements coverage report;
- branches coverage section of detailed test report.

The first report allows to estimate quality of testing in terms of coverage of elementary requirements of the standard. The second one allows to estimate quality of testing in terms of branches of functionality. These reports contain both statistical information about coverage in general and detailed information about coverage each function in particularly.

References

[1] Linux Verification Center (http://www.linuxtesting.org)
[2] *Linux Standard Base Core Specification 3.1*
(http://refspecs.freestandards.org/LSB_3.1.0/LSB-Core-generic/LSB-Core-generic.html)
[3] The UniTesK technology web site (http://www.unitesk.com)
[4] Formal Specification-Based Testing: Getting Started
(http://linuxtesting.org/downloads/getting-started-math-integer.pdf)