

Building a Robust Linux kernel piggybacking The Linux Test Project

Subrata Modak

Linux Technology Centre, IBM, India.

subrata@linux.vnet.ibm.com

Balbir Singh

Linux Technology Centre, IBM, India.

balbir@linux.vnet.ibm.com

Abstract

The LinuxTM kernel is growing at a rapid rate and runs across many architectures and platforms; ensuring that the kernel is reliable, robust and stable is very critical. The Linux Test Project (LTP) was established to meet the very goals stated above. Testing is often ignored in major development and we pay the cost through frequent updates, frequent crashes and unhappy users. LTP is now breathing a new life, we want to add more test cases, cover more code, test new features, update existing test cases and improve the framework.

In this paper, we explore the effectiveness of testing via LTP, and, look at coverage statistics and number of new test cases added. We look at where LTP development and kernel testing via LTP stands w.r.t. kernel development. This paper also demonstrates how to write a simple LTP test case and enjoy the benefits of using it over and over again.

1 Introduction

The Linux Test Project, created by SGI¹, was one of the first to bring organized testing to Linux. No formal testing methodology was available to Linux developers prior to the arrival of LTP. Systematic integration testing was a distant dream though most developers unit-tested their own enhancements and patches. LTP's primary goal is (and still is) to provide a test suite to the Linux community that helps to validate the reliability, robustness and stability of the Linux kernel. It provides functional and regression testing with or without stress, utilizing its own execution harness to allow for test automation.

At the time when 2.3 kernel was released, LTP had around 100 tests [6]. As Linux grew and matured through 2.4, 2.5 & 2.6 kernels, the LTP test suite also

grew and matured as well. Today, the Linux Test Project contains well over 3000 tests, and the number of tests are still growing. It has evolved over years to become much comprehensive, so as to test various features of Kernel including system calls, memory management, inter-process communication, device drivers, I/O, file systems, and networking. With 95% of the test code written in C, today it has become one of the de-facto verification suite used by developers, testers, and Linux distributors who contribute enhancements, bug fixes and new tests back to the project.

2 Breathing a new life into LTP

LTP is now breathing a new life. New test cases have been added, many more test cases have been fixed starting from early 2007 through the early quarter of 2008. Table 1² provides details of test cases that have been added in the mentioned period, and the date as of writing of this paper. Kdump test cases by their very own nature test the Kdump kernel. Similarly the Real Time Linux Test cases are meant for the RT kernel. These additions show that LTP is a part of the daily testing activity of several people involved with kernel testing. It also shows the flexibility that LTP provides to test case writers and executors.

LTP [2] also saw a massive cleanup of the existing test cases. Around 350 Patches were applied, 1000 new sources added, ending up modifying 1000 and removing 247 source files. One of the limitations of LTP was the broken issues, that were preventing further adoption and expansion of LTP, were also addressed effectively.

Some of the issues that were addressed as part of LTP refresh are:

1. The release pattern of file packages were revived to include results on various architectures. With that

²Data as on 15th April 2008, from initial addition to subsequent patching

¹Silicon Graphics Inc.

Name/Type	Total Sources	Avg. Code Size (bytes)
Kdump, Kdump for Network Partition dumps	26	2312
Uts, Sysvipc, & Pid Namespace	27	2614
Inotify	4	5894
Writev	7	7712
Swapon	4	8975
Numa	6	6986
Remap_file_pages	3	6465
Nfs Check Tests	1	1834
Posix_Fadvise & Fadvise64	5	4003
Madvise	4	6572
Sendfile64	7	5625
Arm Specific Test Cases	1	1091
Real Time Linux Test Cases	101	3400
Fallocate	5	7071
Filescaps	11	2579
Cpu Controllers	17	5134
Msgctl	12	7985
Ti-rpc	588	3218

Table 1: Specific List of Test Cases Added to LTP [4]

LTP achieved the release of 169 packages (totaling 265 MB of code), with overall 31458 Packages downloaded, making an average of 65 downloads per day [1].

2. Gcov-kernel patches for Kernels 2.6.18, 2.6.19, 2.6.20, 2.6.21, 2.6.22, 2.6.23, 2.6.24 & 2.6.25 were also made available to the community through LTP
3. Addition of RHEL5 LSPP Test suite Release (EAL4 + Certification Test Suite)
4. Addition of SGI Common Criteria EAL4 certification test suite for RHEL5.1 on SGI Altix 4700 (ia64) and Altix XE (x86_64) Systems.

While LTP worked hard to retain the confidence of the Linux community, it also saw a revamp of the infrastructure (that it provides to run tests) by providing:

- *Output logs in a more attractive/decipherable HTML format.* Figure 1 depicts the new HTML output format for LTP with clear distinction between FAILED, PASSED, WARNED and BROKEN testcases. It is expected that HTML format can be used to show overall test status, and help interactively explore failures. It is also envisioned that using XML in future will allow the results to be validated, converted to attractive formats using style sheets, and many more such advantages of XML can be better exploited.

LTP Output/Log (Report Generated on Mon Mar 31 12:14:29 CDT 2008)

PASSED **FAILED** **WARNING** **BROKEN** **RETIRED** **CONFIG-ERROR**

[Click Here for Detailed Report](#)
[Click Here for Summary Report](#)

Detailed Report

No	Test-Name	Command-Line	Test-Output	Termination-id
1	abort01	ulimit -c 1024:abort01	abort01 1 PASS : Test passed	0
93	accessat01	accessat01	accessat01 0 FAIL : accessat(1) failed, errno=20 : Not a directory	1
94	fallocate01	fallocate01	fallocate01 0 WARN : System doesn't support execution of the test	0
183	truncate04	truncate04	truncate04 1 COMP : The filesystem where /tmp is mounted does not support mandatory locks. Cannot run this test.	0

Summary Report

Test Summary	Can reported some Tests FAIL
LTP Version	LTP-20080331
Start Time	Mon Mar 31 12:14:29 CDT 2008
End Time	Mon Mar 31 13:05:21 CDT 2008
Log Result	/root/subrata/tp/tp-full-20080331/results
Output/Failed Result	/root/subrata/tp/tp-full-20080331/output
Total Tests	860
Total Failures	3
Kernel Version	2.6.21.3
Machine Architecture	i386
Hostname	sniff

Figure 1: A Sample new HTML format for LTP Output

- *Adding discrete sequential run capability.* LTP has an existing option to run the suite for definite period of time, say 24 hrs [7]. The drawback with this approach is that test run can terminate midway without completing the last loop due to time pre-emption. This new feature allows test to execute as many loops as specified by the user, irrespective of the time consumed. Particular test case executed in multiple loops are properly tagged to distinguish outputs generated in multiple loops.
- *Auto Mail Back option of reports.* LTP now pro-

vides the option to collate all outputs & logs, `tar(1)` them and finally mail them back to a specified email address, after entire testing is complete. This can be handy in situations where the tests are run (in background) in remote servers for longer duration of time. On completion the user gets the collated reports in his/her mailbox; a handy indication of completion of tests.

- *Generating default file for failed tests.* LTP now generates a file containing a list of exclusive test cases which *failed* during test run. This file is created in a format which then can be directly used to do a quick re-run of these failed tests. User now can collate the output of only failed tests and debug more efficiently.
- *Integrating better stress generation capability.* LTP employs a parallel infrastructure to create stress (IO/Memory/Storage/Network) on the system, to verify test case behavior under extreme condition(s). The full potential of this infrastructure was not exploited earlier. LTP now provides expanded options to utilize the existing features of stress generation.

In the recent past, there has been focus on running LTP tests concurrently [3]. Several fixes have been provided in this regard to allow tests to run concurrently.

The other area of focus has been to help developers write unit test cases, without the need to download the original LTP-Suite. LTP development rpms for various architectures (i386, x86_64, ia64, ppc64, s390x, etc) are now being regularly released to address this. This is to motivate developers to write unit test case(s) on their own, build them with the LTP development rpms, test them and finally integrate them to mainstream LTP. Intermediate releases are now regular, which gives developers time to fix any build breaks before the final month-end release.

While we aim to increase the Kernel code coverage, we also took a holistic look into the source code that we added to LTP suite during this transition period. The results showed that the LTP code has increased **42%**³ starting 1st January 2007 till 15th April 2008. Though this is quite a small figure compared to what Linux Kernel has grown, the most important thing to note is that

³Data Generated from diffs of ltp-20061222 & ltp-20060415.

the same has been achieved by a very small group of LTP developers.

3 Kernel Code Coverage Statistics

One of the metrics to measure the effectiveness of testing is code coverage [10]. We've run coverage with the gcov patch (linux-2.6.24-gcov.patch⁴) on a x86 system and run different versions of LTP on the same Kernel. However, during code coverage we have not considered Kdump tests, RT tests, DOTS, Open_Posix_Testsuite, Open_HPI_Testsuite, Pounder21 & SELinux testcases. Table 2 shows the code coverage for top 10 items.

Directory	Coverage	
fs	49.8%	10135/20367 lines
include/asm	49.4%	595/1204 lines
include/linux	58.7%	2239/3812 lines
include/net	56.2%	990/1762 lines
ipc	52.8%	1442/2729 lines
kernel	38.2%	9880/25837 lines
lib	42.2%	2105/4992 lines
mm	51.5%	6899/13396 lines
net	65.4%	630/964 lines
security	51.9%	666/1283 lines

Table 2: 2.6.24 kernel code coverage using December 2006 LTP

Directory	Coverage	
fs	52.9%	10778/20367 lines
include/asm	50.9%	613/1204 lines
include/linux	60.0%	2283/3812 lines
include/net	57.6%	1015/1762 lines
ipc	56.4%	1539/2729 lines
kernel	39.1%	10097/25837 lines
lib	43.2%	2159/4992 lines
mm	52.7%	7066/13396 lines
net	65.7%	633/964 lines
security	51.9%	666/1283 lines

Table 3: 2.6.24 kernel code coverage using March 2008 LTP

⁴Available at <http://ltp.cvs.sourceforge.net/ltp/utlils/analysis/gcov-kernel>

The coverage was obtained by running December 2006 LTP against a gcov instrumented 2.6.24 release of the kernel. Table 3 also shows the code coverage for top 10 items. This coverage was obtained by running March 2008 LTP against a gcov instrumented 2.6.24 release of the kernel.

Comparing Table 2 and 3, we make the following observations:

- Between the two runs, the coverage of the recent LTP is better. This is a good sign and is indicative of the progress that LTP has made.

Subsystem	% Increase
Filesystems	3.1
include/asm	1.5
include/linux	1.3
include/net	1.4
ipc	3.6
kernel	0.9
lib	1.0
mm	1.2
net	0.3
security	0

Table 4: Increased coverage due to LTP enhancements between Dec 2006 & March 2008

Table 4 shows the percentage increase in coverage between both the runs.

- Two subsystems *fs* and *include/asm* now have coverage greater than 50 percent
- The data also points us to some interesting facts such as:
 - LTP needs to do a better job of covering the error paths. Some of them need to be covered using the fault injection framework. One limitation of LTP is that the test cases cannot handle faults from the kernel. The test case exits on failure. We propose a new *LTP robust* subproject to allow LTP to work well with fault injection.
 - It is not possible for LTP to cover certain scenarios. With a wide set of permutable config and boot options, it is not possible to test every config/boot option and extract coverage.

We've tested the most common and minimal configuration that works on our machine.

- It is not possible for LTP to handle coverage of code that is not exposed to user space. For example, a machine may be configured with SPARSEMEM, FLATMEM or DISCONTIGMEM. Testing these options and obtaining coverage data is not possible.
- There are several areas of code that have no coverage. We've taken up those areas as areas that need more test cases. Section 7 provides more details about our future plans.

- We intend to make code coverage data available to the LTP website⁵, so that developers can see how well their code is tested. This might even motivate them to contribute the test cases they've used for testing the feature to LTP.

4 Role of LTP in testing Linux

Software testing can be broadly categorized into

- Compilation Testing⁶
- Unit Testing
- Functional Testing
- System Testing
- Stress Testing
- Performance Testing

LTP helps with *Functional*, *System* and *Stress* testing. LTP cannot directly do *Compilation*, *Performance* or *Unit* testing.

There are several ways of testing the Linux kernel. Most developers run the latest kernel on their desktops and servers. The kernel gets tested via the applications that get executed. Any major performance regression is observed and reported.

⁵<http://ltp.sourceforge.net/>

⁶Many textbooks on software testing, do not include build as a part of the test effort. Since Linux runs on several platforms and has several features that can be enabled/disabled at compile time, ensuring that the build works well across the platforms, architectures and features is an important aspect of testing Linux

LTP goes a step further by providing test cases that test user interfaces with several valid and invalid parameters. It tests various subsystems of the kernel such as the memory management code, the scheduler, system calls, file systems, real time features, POSIX semantics, networking, resource management, containers, IPC, security, timer subsystem and much more. LTP provides an infrastructure to stress test the system by

- Providing test cases that stress the system
- Allowing concurrent execution of test cases
- Providing noise in the background (CPU, Memory, Storage, Network, etc) while running tests

LTP plays an important role in system testing. Several users of LTP use it to validate their entire system. Running LTP validates the "C" library and the user interface(s) provided by the kernel (to the extent test cases have been added).

LTP is also run by kernel testers for regression testing. Given the size and nature of the LTP test cases, it provides a good framework for executing desired tests, selecting a subset of those tests as basic acceptance test, and running them.

In future, we intend to enhance LTP to provide facilities for performance testing⁷ and more test cases that can test the functionality of features not yet in the mainline Linux kernel. This would help provide extensive testing of a feature before it gets into the mainline Linux kernel.

5 Early and Effective Testing

Up to a point it is better to let the snags [bugs] be there than to spend such time in design that there are none (how many decades would this course take?)

A M Turing, Proposals for ACE (1945)

The importance and significance of effective and early testing cannot be stressed enough. According to Barry Boehm's Software Engineering Economics [5], time required to identify a defect in software after it has been deployed is 40 to 1000 times longer than if had been found in the requirements analysis. While testing cannot really catch bugs introduced in the requirements phase,

⁷By providing a performance testing framework

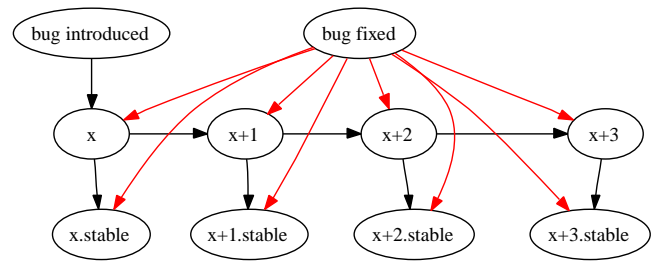


Figure 2: Sample bug fix flow for a bug introduced in version x and fixed in version $x+3$

it certainly can help catch them before the code is deployed.

Consider a hypothetical example of a feature that introduces a bug into version x of the kernel. The bug is tested and detected in version $x+3$. Figure 2 shows in "red" color the versions into which the bug needs to be fixed. If Linux distributions have spawned off kernels in between versions x and $x+3$, then more bug fixing, hot fixes and updates need to take place.

The example scenario above shows the advantage of early and effective bug fixing. Had the bug been detected in version x itself, the unnecessary overhead of bug-fixing, maintenance and additional testing could have been avoided.

This brings up an important question - To what extent do we test the kernel? We believe that all bugs that can be caught easily and with some effort and observation must be discovered and fixed. Turing's quote at the beginning of the section refers to a good trade off between bugs and time spent.

6 Simplest way to write a LTP test case

There has been papers written by LTP Maintainers/developers regarding ways/methodology to write a simple LTP test case. Notably amongst them are:

- Testing Linux with the Linux Test Project [9], and
- Improving the Linux Test Project with Kernel Code Coverage Analysis [8]

All of these are easily available in archives, hence we skip the intricate details of writing a testcase. We instead focus on presenting a set of workflows to depict the overall mechanism to run the LTP suite, and individual test case execution.

6.1 LTP Suite Execution Framework

Figure 3 depicts the flow of how the entire LTP suite works. On invocation, *runltp* script parses all options. It proceeds to generate the list of test cases to be executed depending on user choice at command line. It exports all those identifiers necessary for test execution next. Optionally, it can also generate certain stress on the system. Next it invokes the test driver PAN, which then takes care of executing each test case in the list. Once all test cases are executed, PAN reports PASS if all test cases have executed successfully. Else, it returns fail if at least one of them failed. Generation of HTML output and auto-mail-back is optional. Once that is over, the script does the necessary cleanups (releasing resources, clearing system stress, etc) and exits.

6.2 Individual Test Execution Framework

Figure 4 shows the preamble, which starts with mentioning the copyright statement(s) followed by the GPLv2 declaration (which is mandatory). Following that, the test case name and algorithm are described. Modification history is maintained to identify sources of this code modification: the author, date and reason of modification.

Figure 5 shows the the main body of the test case. It starts by including headers, declaring general and LTP specific global & static identifiers. Once inside the *main()* block, the first thing is to check whether the corresponding feature, that this test case is supposed to test, is supported by this kernel version/architecture/FS type/glibc version. If any of these evaluates to false, the test is aborted, corresponding message written to logs/output, cleanups done and test exits with proper exit-value. If everything is supported, the main code of testing is executed.

If there are some BROKEN or WARNING messages generated, then the test takes appropriate action. Assuming everything goes well, test execution status is written to log/output, cleanups are done, and the test exits with a proper return value.

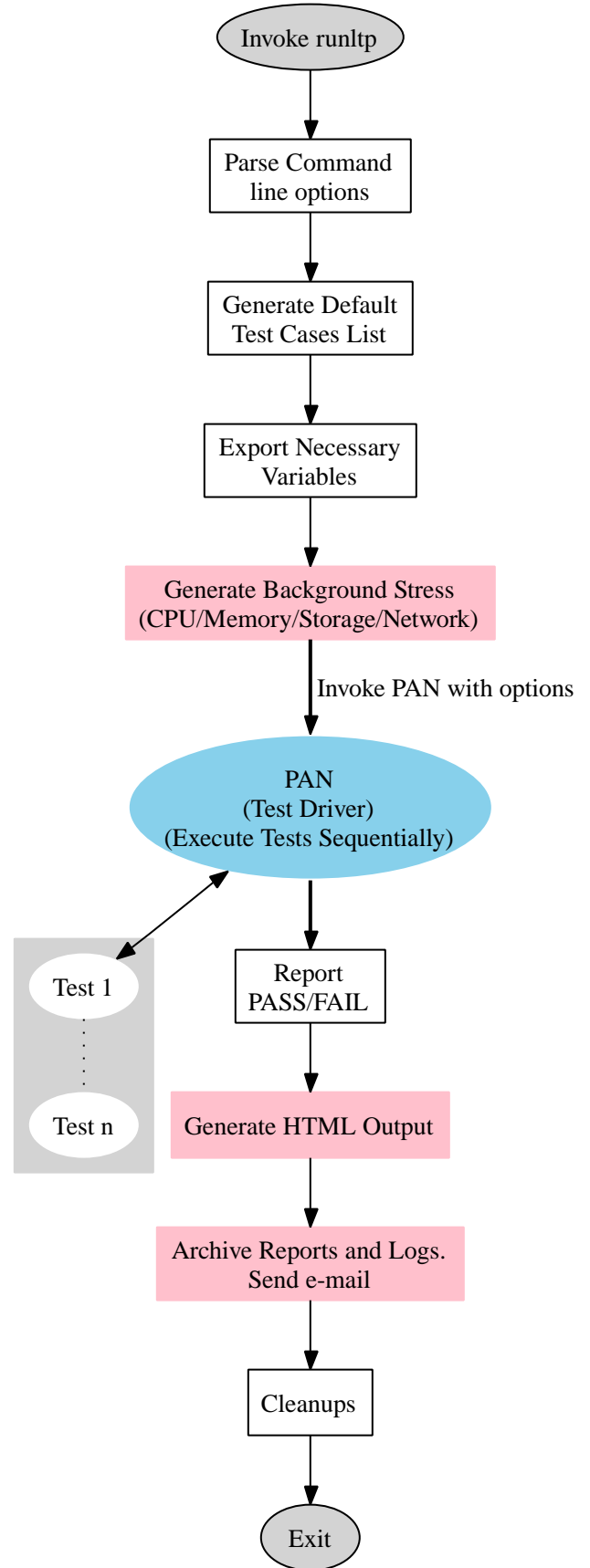


Figure 3: LTP Flow Diagram

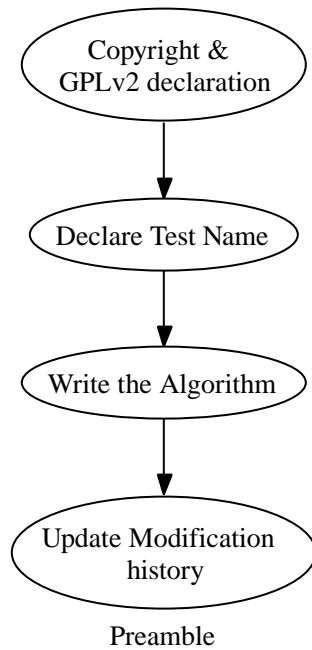


Figure 4: LTP Test Flow Preamble

7 Future Plans

Several initiatives has been taken so far to improve LTP. Most of them has been successful. We plan to take up more such initiatives in future. As a part of LTP's initiative to involve Kernel developers in particular, we have already started with the concept of LTP development packages. These are a combination of LTP-specific libraries, header files, executables and manpages, easing developers' task of developing unit test cases, for the features that they plan to merge to the Kernel.

Another initiative is starting the LTP-*mm* tree. Developers might not wait for their features to be part of mainline Kernel and then open up the test cases. The same test cases can be contributed to the LTP-*mm*. Test cases can be contributed to the LTP-*mm* project as early as the corresponding feature hits any Kernel tree (mm,rc,etc), or planning to get into any tree. The test case(s) themselves can be modified multiple times in resemblance to the corresponding feature changes/modification in the Kernel tree. Once the feature becomes part of mainline Kernel, the corresponding test cases are moved from LTP-*mm* project to main LTP project.

While we will be happy to have those test cases use the LTP-specific logging libraries, it is not a mandatory requirement. If the test case(s) is/are written in C/Shell

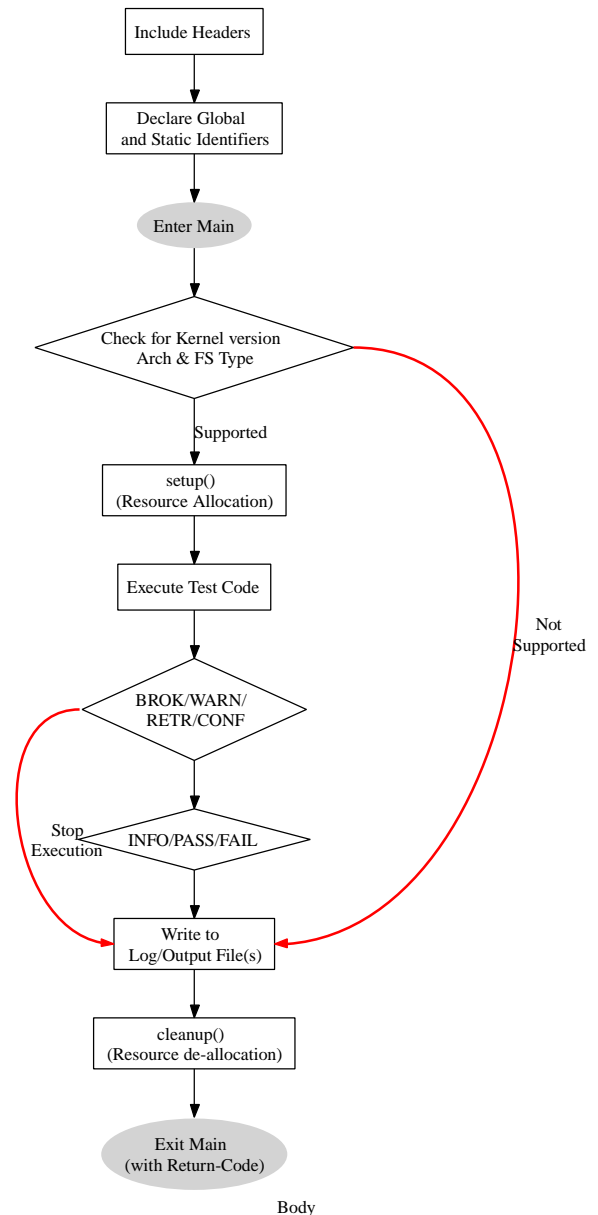


Figure 5: Individual Test Flow Diagram

and returns 0/1 on PASS/FAIL, then it is a very good candidate for inclusion into the LTP. We encourage Kernel developers to contribute their unit test cases in whatever form they have. LTP community will help them in converting them to the required format across time. We would also urge test cases to find their way to the LTP in many unexplored areas, like the device drivers, and also in areas where the **kernel code coverage is very low**.

When we request the community to contribute to test case(s) development, we also want to convey to you that the LTP aims to include new test cases in the areas of:

SAMPLE LTP OUTPUT

```
<<<test_start>>>
tag=remap_file_pages01 stime=1208361993
cmdline="remap_file_pages01"
contacts=""
analysis=exit
initiation_status="ok"
<<<test_output>>>
remap_file_pages01
1 PASS : Non-Linear shm file OK

<<<execution_status>>>
duration=1 termination_type=exited
termination_id=0 corefile=no
cutime=7 cstime=2
<<<test_end>>>
<<<test_start>>>
tag=faccessat01 stime=1208362004
cmdline="faccessat01"
contacts=""
analysis=exit
initiation_status="ok"
<<<test_output>>>
faccessat01
6 FAIL : faccessat() Failed, errno=20 : Not a directory

<<<execution_status>>>
duration=1 termination_type=exited
termination_id=0 corefile=no
cutime=8 cstime=2
<<<test_end>>>
<<<test_start>>>
tag=fallocate01 stime=1208363009
cmdline="fallocate01"
contacts=""
analysis=exit
initiation_status="ok"
<<<test_output>>>
fallocate03
0 WARN : System doesn't support execution of the test

<<<execution_status>>>
duration=1 termination_type=exited
termination_id=0 corefile=no
cutime=8 cstime=2
<<<test_end>>>
```

Figure 6: Sample LTP Output

- Power Management testing,
- Controllers and Containers testing,
- KDUMP (kdump on Xen hypervisor and guests),
- Union Mount,

SAMPLE LTP LOG

```
Test Start Time: Wed Apr 16 21:47:41 2008
-----
Testcase          Result          Exit Value
-----
remap_file_pages01  PASS           0
faccessat01        FAIL           1
fallocate03         WARN           1
-----
Total Tests: 2
Total Failures: 0
Kernel Version: 2.6.18-53.1.13.el5
Machine Architecture: i686
Hostname: <sniff>
```

Figure 7: Sample LTP Log

- Sharedsubtree, etc

in very near future. We will continue to fix and improve upon the existing testcases. Enhancements in the area of LTP infrastructure:

- *Development of XML logs/output.* We plan to generate the logs/output in XML format so that they can be parsed easily by any XML parser.
- *.config File based Execution.* Options to run LTP are growing. It may be difficult for users to remember and mention all options at command line. We plan to provide .config file, which will host all such options in *variable=value* format. Users will be required to just run *runltp*, which will automatically parse options from the .config file.
- *Network based installation, execution and report collection.* We plan to create an infrastructure where it will be possible to provision LTP on multiple machines from a central machine. The provisioning server will be capable of deploying LTP suite across multiple machines, build & install, run, and bring back all of the reports.

will also be forthcoming.

Recently we identified that many LTP test cases fail while running concurrently. We plan to make entire LTP suite **concurrency-safe** [3]. We also plan to test all Kernel releases(*mm, rc, etc*) and make those results available on the LTP website along with the code coverage

details against the latest stable Kernel. In near future, LTP will continue to focus on all possible means to improve code coverage. However in long term we would also consider adding **benchmark** infrastructure to LTP.

8 Conclusion

As Linux testing evolved through the ages, other test projects/suites with better infrastructure/features came to the center-stage. LTP is considered of having the following defects: lacking the facility to provide automatic Kernel build/reboot and test, having low code coverage, non-parsible output logs and broken test cases. While it is true that the LTP does not provide autobuild and test options, that was not the main focus area. It was designed to be a very handy regression test suite. LTP along with other test suite(s) can complement each other's features to **implement the much broader goal of making Linux better**.

The *Kernel code coverage cannot be drastically improved without the corresponding test cases for Kernel features being made available to the LTP*. While this **responsibility cannot be enforced**, the impact of such can be brought to developers notice. Each section of logs/output are properly tagged, which in turn can be parsed by even a simple parser. **Figure 6** depicts a sample test output, and, **Figure 7** shows a sample log file contents. The HTML output depicted in Figure 1 is testimony to the fact that LTP log/output can be parsed very easily. Hence, a simple parser was able to generate this HTML from a normal text output. And this very trivial output format will also enable us to write the future XML parser.

Many test cases were found to be broken, as many Kernel features have undergone changes, and, the same testcases were not cleaned. LTP clearly distinguishes the way test cases should report results, with keywords like INFO, BROK, CONF, RETR, PASS, FAIL, etc well documented in the LTP manpages. Elaborate information about the test case behavior is also reported along with these keywords.

Everybody can put forth their views of how LTP should move forward, what it should address, and what it should avoid. The LTP community highly appreciates patches, the benefit of which goes directly to all. LTP has discovered opportunity in positive criticism, and has focused with more vigor on it's primary goal of providing a functional and regression test suite. It will

keep growing at any cost along with the growing Kernel, while simultaneously addressing bottlenecks in other areas too. **However, we need more active contribution from the Kernel developers to make LTP a very strong & useful Test Suite for the Linux Kernel.**

Acknowledgments

We would like to thank Robert Williamson, IBM, for his input to and review of drafts of this paper. We also owe lot of thanks to Sudarshan Rao, Premalatha Nair and our team mates for their active support and enthusiasm. And a special thanks to all those LTP developers whose immense contribution keeps this project growing.

Legal Statement

©International Business Machines Corporation 2008. Permission to redistribute in accordance with Linux Symposium submission guidelines is granted; all other rights reserved.

This work represents the view of the authors and does not necessarily represent the view of IBM.

IBM, IBM logo, ibm.com, and WebSphere, are trademarks of International Business Machines Corporation in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.

INTERNATIONAL BUSINESSMACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you. This information could include technical inaccuracies or typographical errors. Changes

are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

References

- [1] Linux test project download page. https://sourceforge.net/project/showfiles.php?group_id=3382.
- [2] Linux test project home page. <https://sourceforge.net/projects/ltp>.
- [3] Linux test project mailing list. https://sourceforge.net/mailarchive/forum.php?forum_name=ltp-list.
- [4] Linux test project source code repository. <http://ltp.cvs.sourceforge.net/ltp/>.
- [5] B. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.
- [6] N. Hinds. Kernel korner: The linux test project: Finding 500 bugs in 50 different kernel versions is the fruit of this thorough linux testing and code coverage project. *Linux Journal*, 2004. <http://www.linuxjournal.com/article/7445>.
- [7] M. Iyer. Linux test project documentation howto. <http://ltp.sourceforge.net/documentation/how-to/ltp.php>.
- [8] P. Larson. Improving the linux test project with kernel code coverage analysis. Linux Symposium 2003.
- [9] P. Larson. Testing linux with linux test project. Linux Symposium 2002.
- [10] P. Larson, R. Williamson, and M. Ridgeway. Linux test project technical papers. <http://ltp.sourceforge.net/documentation/technicalpapers>.