Proceedings of the Linux Symposium

Volume One

July 20nd–23th, 2005
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*
Stephanie Donovan, *Linux Symposium*

Review Committee

Gerrit Huizenga, *IBM*
Matthew Wilcox, *HP*
Dirk Hohndel, *Intel*
Val Henson, *Sun Microsystems*
Jamal Hadi Salimi, *Znyx*
Matt Domsch, *Dell*
Andrew Hutton, *Steamballoon, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*

Usage of Virtualized GNU/Linux for Binary Testing Across Multiple Distributions

Gordon McFadden
Intel corporation
gordon.mcfadden@intel.com

Michael Leibowitz
Intel Corporation
michael.leibowitz@intel.com

Abstract

In this paper, we will discuss how we created a test environment using a single high-end test host that implemented multiple test hosts. The test environment enabled the testing of software running on different Linux distributions with different kernel versions. This approach improved test automation, avoided capital expenditures and saved on desktop real-estate. We employed a version of Gentoo Linux with a modified 2.6 kernel, along with multiple instances of different distributions and version of Linux running on User Mode Linux (UML). The particular tests involved are related to the Linux Standards Base, but the concept is applicable to many different environments.

We will describe how we improved aspects of the Gentoo kernel to improve performance. We will describe the methods used to affect a lightweight inter UML communications mechanism. We will also talk about the file systems chosen for both the host OS and the UML. Finally, we will have a brief discussion around the benefits and limitations of this type of test environment, and will discuss plans for future test environments.

1 Introduction

While setting up a test environment to execute tests to verify compliance of various Linux distributions as part of the testing of the Linux Standard Base (LSB) [1] 3.0 Specification, it was noted that both time and capital expense could be saved if the host running the tests could be effectively reused.

In the context of executing LSB conformance tests, it is the cass that many of the tests can be readily executed in an automated and autonoumous manner. Only some tests are manual in nature and require the attendance of a test operator. It was also noted that the tests require different distributions, including different kernel versions.

Given the fact that the test cycle was not expected to last indefinitely, and that the number of distributions under test was likely to increase, it did not make sense to attempt to allocate one host to each distribution.

Additionally, it was important to the test philosophy that the distributions be available at all times, allowing tests to run independently of each other. If a multi-boot system, such as GRUB or LILO were employed, then testing could only proceed sequentially.

Another required aspect of the test environment

is the ability to instantiate tests without impacting other running tests.

The solution that is employed is the use of a host operating system running Guest Operating Systems (GOS) in User Mode Linux (UML) [2]. Gentoo [3] release 2.6.11-R-6 was chosen as the host operating system because it is very configurable in the areas of file systems, how many process are running and other areas. The intent is to keep the host of installed software on the host operating system very small. It is very easy to install a minimal set of packages in a GenToo build. While any distribution provides the ability to configure installed packages, and allows modifications to the kernel, the GenToo distributions seems to be geared toward allowing installers to make the types of modifications needed to encompass the solution.

It makes a great deal of sense from the perspective of cost and space to arrange the test environment to use one host per architecture.

1.1 Changes to the kernel

1.1.1 Elevators

The processes of optimizing the kernel started with the 2.6.11.6-vanilla Linux Kernel and involved modifications to the elevator to increase performance of spawning UML instances. The Linux kernel implements a disk I/O scheduling system referred to as the elevator. The name elevator comes from the conceptual model of the disk drive as a linear array with a single read/write head. The head moves up and down the disk, as an elevator moves in an elevator shaft, and the blocks that are read or written to as the call buttons on various floors. As in the real world, the algorithm for moving the elevator in response to floor requests is a non-trivial dining philosophers type of problem. Responsiveness, repeatability, equity, and aggregate bandwidth must all be carefully balanced. No algorithm can always maximize all of these needs, but any suitable algorithm should be able to avoid starvation in all circumstances. User Mode Linux also has an elevator, which operates in the same way as the host elevator does. Because several elevators may be in use on multiple encapsulated operating systems in parallel (see Figure 1), they can effectively "collude" to starve one or more processes of disk access. The elevator of the host kernel was modified to better deal with this situation.

1.2 File System Issues

XFS was chosen as the host file system. When XFS was devised by SGI, it was designed to be able to give high throughput for media applications. Filesystem-within-filesystem applications are similar to media, in that both involve contiguous large files that are accessed in regular ways. In non-linear editing applications, files are written to and read not in a strictly linear fashion, but in large linear blocks. File system access from an guest operating system in similar due to the elevator inside the encapsulated kernel.

1.3 Execution Environment

There are two modes of executing kernels in a UML environment. The first is refered to as Tracing Threads (TT) mode. The second mode is Separate Kernel Address Space (SKAS) mode. In the TT mode, the processes and the kernel of the GOS all exist in the user space of the host kernel. In SKAS mode, the kernel is mapped into its own address space. The advantage to Tracing Thread mode is that there is support for Symetric Multi-Processor

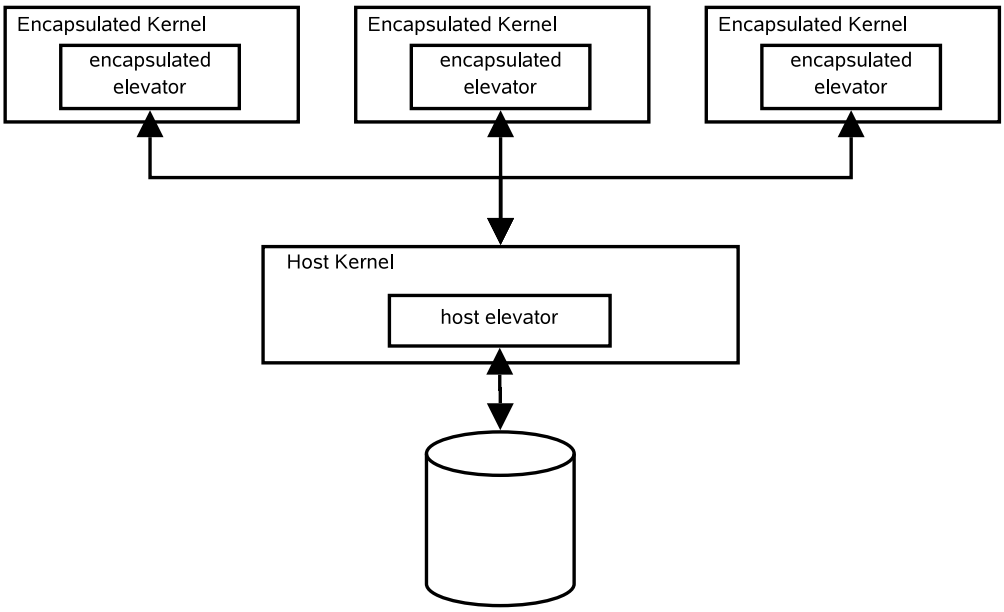

Figure 1: Nested Elevators

(SMP) based platforms. The most compelling reason to consider using the SKAS mode is the perfomance advatage it holds over TT mode. This advanage is most noticable in applications that are fork() intensive.

To work in SKAS mode requires a minor patch to the host OS kernel. This patch was examined and it was determined that for the purposes of the specific LSB tests being considered, the patch did not affect the viability of the test.

It was not nessesary to have SMP available to the GOS kernels in order to run the tests, and the host kernel effectively makes use of the SMP platform. Therefore, the desision was made to employ the SKAS patch.

1.4 Distributions Tested

The following represents a sample of the distributions required for the LSB 3.0 testing:

- Novell Linux Distribution 10
- RedHat Enterprise Linux 3
- RedHat Enterprise Linux 4
- Red Flag

1.5 Intra-UML communications

In the deployed environment, even though each GOS has it own IP address and stack and is connecte via a virtual switch, there was no requirement for communications between individual GOSs. In the future, it is foreseen that extending the test environment to allow Client/Server style tests on separate GOSs may provide value. In this environment, GOSs could communicate in one of three methods. First of all, given the fact that there is an IP stack running on each GOS, then socket based communications are available. This would include direct sockets, ssh, ftp, rsh, and other well known IP-based communications methods. Second, it would also be possible to use semaphore files between GOSs in a manner similar to that described in this paper. Finally, it is theoretically possible

to attach TTY/PTY devices between GOSs, allowing character based traffic to be passed between two GOSs. This approach would have very little overhead, and may be very attractive as a management conduit for test control. More research is needed in this area.

2 Concurrent Test Limitations

It is important to understand the limitations that exist when running encapsulated or virtualized test environments. These limitations associated with running concurrent tests on a UML based system include:

- hardware abstraction—it may not be appropriate to test the hardware and hardware abstraction layers since some aspects of the encapsulated operating systems are abstraced. Example of this are the apparent memory size of which the encapsulated system is aware, the block I/O systems, etc.
- resource sharing—it is possible for a test to have different resources available for different invocations of the test. This may produce different results in the area of execution time, CPU utilization, and other similar measurements.
- inter-client communication—it adds value to the test cycle for the host operating system to be able to communicate with the guest operating systems for the purpose of kicking off tests and recovering test results. In the future it may also be useful to enable communication between encapsulated systems.

For the purposes of the LSB testing, these limitations are not onerous, and the test environment is sensible.

3 The Cost of UML

As with any system of emulation, encapsulation, or virtualization, there is some performance penalty to be expected. Because the LSB compliance testing takes such a long time to execute, two synthetic benchmarks were chosen that isolate particular areas of system performance that have a large impact on our tests. Of principal interest is file system performance and scheduler performance.

3.1 File System Test results

One of the benchmark tools employed was Bonnie++ (1.03a) a widely accepted file system throughput test. The Bonnie++ benchmark was used the in the development of the ReiserFS file system. Even though a HyperThreaded machine is used as the test host, it was decided to use single threaded mode for bonnie++ because the primary interest is in the performance that *one* process would receive, rather than trying to approximate a full running system in some way. Bonnie++ was configured to choose the set size, which for the host was 2G. The encapsulated operating systems have varying quantities of memory, so the same set size as the host was not used. The intent of this study was to compare file system performance in the encapsulated operating systems, rather than comparing encapsulated performance to the host. See Table 1 for details on file system throughput.

3.2 Scheduler Performance

Because the LSB tests require a large number of sequential and concurrent operations, scheduler performance inside the encapsulated operating systems is of interest. There are two factors of interest here, the process creation overhead and the context switching time. To measure the former, the `spawn` test program of the

	Sequential Output			Sequential Input	
	Per Char	Block	Rewrite	Per Char	Block
Host Kernel	KB/s	KB/s	KB/s	KB/s	KB/s
2.6.11.6-skas3-v9-pre1	30775	64928	22433	14864	53974
2.6.11.6	30726	65639	22906	15159	54649
	.16%	–1.08%	–2.06%	–1.95%	–1.24%

Table 1: Host disk throughput comparison

unixbench-4.1.0 test suite was used. To measure the latter, the context switching measurements of the lmbench-3.0-a4 test suite was observed.

4 Testing on UML

One of the factors that influenced the design of the test environment was the relative execution time of the tests in questions. Generally, the compliance tests take in the order of an hour to execute. The Application Battery suite of tests for LSB certification takes in the order of 3 hours per distribution, and is a very manual operation. The validation of the Sample Implementation takes about 30 minutes. The full testing of the distribution using the runtime library is documented to take approximately seven hours on a uni-processor host.

Since the tests take so long to execute, there is no need to launch the tests instantly. If it takes one or two seconds to cause the testing to begin, this will cause no appreciable difference in overall test execution time. Accordingly, a file based system was developed based on an NFS file system. Each GOS exports a directory to be used for testing. The host OS mounts a directory for each GOS. The fact that test take a long time to execute also means that that the residual files will persist for quite some time. It is for this reason that `.ini` and `.fini` files are used in the scripts to indicate when a test is ready to start, and when it has completed. This approach also allows multiple tests to be run concurrently.

The appropriateness of running concurrent tests must be determined by examining the many aspects associated with backgrounding tests. It would be possible for one GOS to over consume CPU and disk resources by instantiating many tests. The GOS side of the test environment does not put any restrictions on the number of concurrent tests that may be run.

For a general purpose test environment, it would not be difficult to modify the scripts so they kept track of the number of outstanding test—those actively executing tests—and throttle the arrival rate of tests according to some high water mark.

4.1 Launching a test

Table 2 describes the steps taken by the host OS to launch a test on GOS 2.

For the purposes of this example, it is assumed the test to be run exists in local directory `/opt/test1` in the form of an executable and some supporting files. The tests are to be executed on GOS #2.

Note that the executable test and any supporting data must be transferred before the `.ini` file used to kick off the test is created.

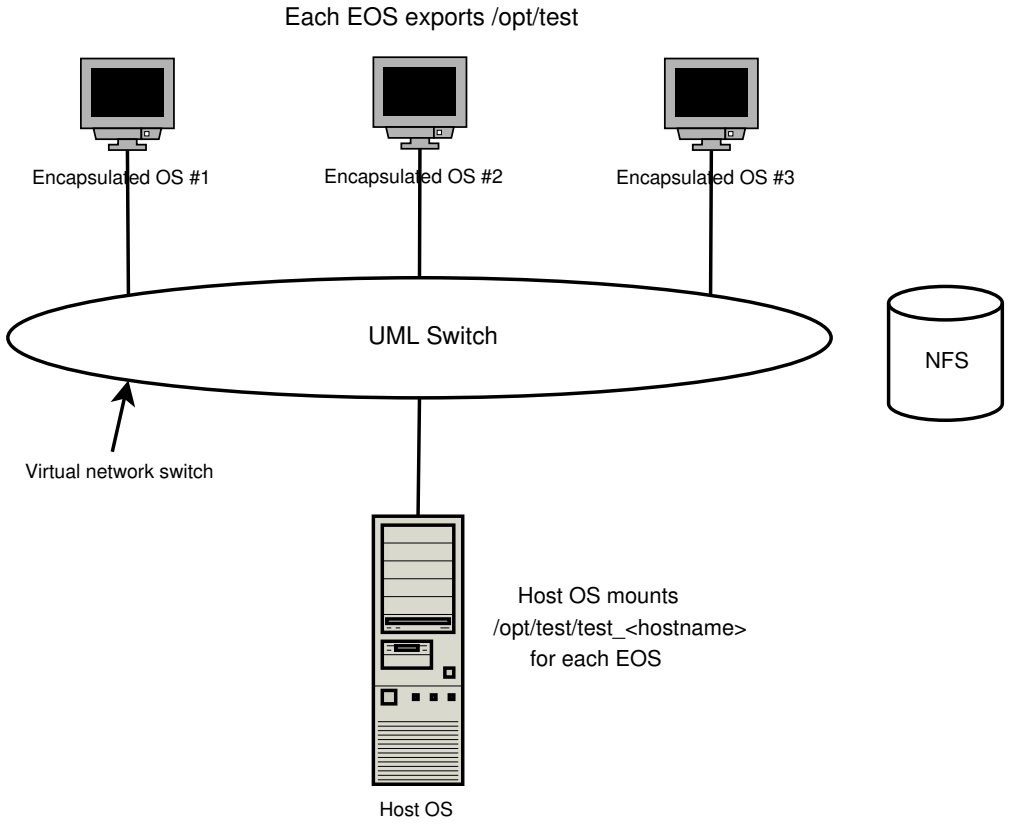

Figure 2: UML Figure

```
 $ host: ls /opt/test1
CVS bin result data

 $ host: ls /mnt/GOS2
 $ host: mkdir /mnt/GOS2/bin
 $ host: mkdir /mnt/GOS2/results
 $ host: mkdir /mnt/GOS2/data
 $ host: cp -R /opt/test1/data /mnt/GOS2/data
 $ host: cp /opt/test1/bin/test_exec /mnt/GOS2/bin/test_exec
 $ host: touch /mnt/GOS2/bin/test_exec.ini
```

Table 2: Script Used to Launch Tests

4.2 Test execution

On each GOS, there is a script running that periodically checks for the existence of a test. The script is presented in Table 3.

If can be noted that this script tests for the existences of `.ini` files in the `.bin` directory, executes the tests redirecting stdin and stderr to files based on the test name in a results directory. When the test has completed, the script create a `.fini` file, which is a flag to indicate the test is complete.

4.3 Getting results

Obtaining the results of the test are reasonably trivial. The test application on the host OS waits for the creation of a `.fini` file in the results directory. Once this empty file is created, then the stdout and stderr of the test can be evaluated to determine the success of failure of the test. If the test generates any log files, then these too can be evaluated.

5 Futures

One of the major drawbacks of the method employed in the test environment described in this paper is the fact that the system resources are not protected. The memory associated with one encapsulated may be partially swapped out in the host operating system. Disk file systems need to be carefully planned, and can not change with out adversely affecting disk performance. The overhead of the host OS also reduces the resources available to the encapsulating OSes.

A solution that addresses the shortcoming is the newly released Virtualization Technology (VT) platform. This platform would a allowed a much faster deployment of the test environment. It also has a much faster transition between guest operating systems due to the hardware assisted switching. A VT platform also allows the operating systems that run on them to have protected hardware resources.

Although not available for this test environment, UML is being ported to a VT technology platform, allowing a more efficient use of system resources from within an GOS. UML is also being ported to x86-64 architures, as well as PPC and S390 processors.

As the testing for the Linux Standards Base continues, it is fully anticipated that this test environment will be migrated to a VT-enabled platform in the very near term.

6 Conclusion

The test environment described in this paper was designed to facilitate simultaneous or near simultaneous testing of different distributions. The nature of the testing involved was well suited for the type of environment available from a UML based test platform. Performance slowdowns were not an issue.

7 Acknowledgments

The authors would like to express their thanks to Jeffery Dike, one of the developers and maintainers of UML, for taking the time to answer questions and for providing information on the future of UML.

```
#!/bin/sh

export TD=/mnt/test
while [ 1 -gt 0 ] ; do
  for test in ‘ls $TD/bin/*.ini‘ ; do
    echo TEST is $test
    export fex=‘basename $test .ini‘
    echo $fex
    if [ -x $TD/bin/$fex ] ; then
      ($TD/bin/$fex > $TD/results/$fex.out 2> $TD/results/$fex.err;
        touch $TD/results/$fex.fini;
        rm $TD/bin/$fex) &
    fi
    rm $test
  done
  sleep 5
done
```

Table 3: Script Used to Execute Tests

8 References

[1] Linux Standard Base at
`http://www.linuxbase.org/`

[2] Read more about User Mode Linux at
`http://usermodelinux.org/`

[3] Read more about Gentoo at
`http://www.gentoo.org/`