NFS server in Linux: Past, Present and Future

Hongjiu Lu
hjl@valinux.com
VA Linux Systems
Sunnyvale, CA, USA

August 15, 1999

Abstract

In this paper, we discuss the NFS server implementation in Linux. We exam the history of NFS server in Linux. We describe the implementation a Linux-based NFS server solution for enterprise, focusing on interoperability and performance measurement.

1 Introduction

The Network Filesystem (NFS) protocol provides transparent remote access to shared files across networks. The NFS protocol is designed to be portable across different machines, operating systems, network architectures, and transport protocols. Remote Procedure Call (RPC) is used to implement the NFS protocol.

The NFS protocol was designed to be stateless. To deal with file and record locking, 2 separate protocols are specified, the Network Lock Manager (NLM) Protocol and the Network Status Monitor (NSM) Protocol. NLM provides advisory file and record locking. NSM provides applications with information on the status of network hosts.

Currently, there are 2 versions of the NFS protocol in use, Version 2 (V2) and Version 3 (V3). This paper investigated the Linux server implementations of the NFS V2 protocol using UDP protocol.

2 Overview

There are 2 NFS server implementations under Linux. One is the user-space NFS server. It was originally developed by Mark Shand, Donald J. Becker, Rick Sladkey, Orest Zborowski and Fred N. van Kempen. It has been maintained and improved by Olaf Kirch. At the same time, Olaf Kirch also was working on the kernel-space NFS server implementation for Linux.

2.1 The User-Space NFS Server

The Linux user-space NFS server is relatively easy to implement. It only uses the user-space RPC calls. It has several features:

- The directory hierarchy exported by the user-space NFS server mount points within an exported filesystem transparently; thus the client sees the same directory hierarchy as is seen from the server.
- Since the server is running in the user space, it is easy to do uid and gid mapping as well as access DNS, NIS, NIS+, etc.

However the user-space NFS server has some drawbacks:

- Since it is running in the user space, it has to do extra memory copy between user and kernel spaces. Also there is an overhead for context switch. They can be very expensive.
- There are no supports for NLM nor NSM. File and record locks are not available.
- The current implementation is not multithreaded which impacts its performance.
- Since the export control is initialized at the startup time, the server process has to be restarted when /etc/exports is changed.

2.2 The Kernel-Space NFS Server

Mainly due to the performance and interoperation considerations, the kernel-space NFS server is preferred over the user-space one. It has many advantages:

- Since it is running in the kernel space and doing RPC calls inside kernel, it doesn't have to move memory between kernel and user spaces. Its memory operation is much more efficient. There is no overhead for context switch.
- Since it is a kernel process, it can take advantage of the kernel thread.
- Both NLM and NSM protocols are supported. File and record locks can work across NFS. It is very important when you have a heterogeneous NFS environment.
- The file export control is implemented as a system call. When /etc/exports is changed, there is no need to restart the NFS server if there are no changes to mountd.

However, there are some disadvantages for the kernel-space NFS server:

- Unlink the user-space NFS sever, each entry in /etc/exports can export only a directory in one file system. Any file systems mounted under the exported directory are not exported. A separate entry is needed for each mounted file system.
- Since it is not easy to do callback from the kernel, uid/gid mapping and other functionalities which require access to DNS, NIS, NIS+ and other user-space services are not supported.

3 Interoperability

To deploy Linux as an NFS server in heterogeneous multivendor environment, we have to ensure that the Linux NFS server can interoperate with different NFS client implementations. We used the kernel-space NFS server in Linux kernel 2.2.9 and the user-space NFS server of version 2.2beta37 to test interoperability of the Linux NFS server implementations.

3.1 File Handle

The definitions of file and file handle are in the NFS V2 protocol are vague. The V2 specification only says a file handle is opaque to client and can contain whatever information the server needs to distinguish an individual file. However, there are many ambiguities:

- If 2 file handles from the same server are identical, can clients assume they represent the same file on the server?
- If 2 file handles from the same server are different, can clients assume they represent the different files on the server?
- Should all the hard links be represented by the same file handle?
- When a client renames a file within the same file system, should it expect the file handle is unchanged?

The V2 specification is not clear on them and different NFS implementations interpreted them differently. That leads to the interoperability problems among different NFS implementations.

For the interoperability test, we used the Connectathon, which is a network proving ground sponsored by Sun Microsystems for testing software and hardware interoperability allowing vendors to test their interoperability solutions, with special emphasis on NFS and Internet protocols, testsuites, http://www.connectathon.org/nfstests.html, to test the Linux implementation of the NFS V2 Protocol. Solaris 7/x86 and Solaris 7/Sparc are used at the NFS clients. We tested both the user-space and kernel-space NFS servers.

As expected, the user-space NFS server didn't support file lock over NFS. All NFS lock tests in the Connectathon testsuites were not performed. Although the rest of tests were ok, the user-space NFS server is not suitable in an environment where the file lock over NFS is required. The later part of this paper will show the performance of the user-space NFS server is not idea either.

3.2 The Kernel-Space NFS Server

The Connectathon testsuites revealed several flaws in the NFS server implementation in the Linux kernel 2.2.9. The test result from Solaris clients showed that the Solaris NFS client assumes:

- Files are unstructured streams of uninterpreted bytes.
- If 2 file handles from the same server are identical, they represent the same file on the server.

- If 2 file handles from the same server are different, they must represent the different files on the server.
- All the hard links should be represented by the same file handle
- When a client renames a file within the same file system, the file handle should be unchanged.

However, two fields in the file handle in the Linux NFS server, fb_dentry and fb_dirino, make Solaris's assumption untrue.

fb_dentry is for dentry, which is the internal part for the Linux file system. It determined by pathname. That implies several things:

- When a pathname is deleted and then created, their dentries will be the same although they are different files.
- The hard links have different dentries.
- Rename will lead to different dentries.

Those make dentry very unsuitable for file handle. Even worse, dentry is a memory pointer. That means the file handle from the Linux NFS server is not persistent. When the server is rebooted, the file handlers which clients have may become stale and for the same file the Linux NFS server may return a different file handle.

3.3 File Handle in Kernel-Space NFS Server

To make the file handle from the NFS server in the Linux kernel to work with other NFS client implementations, we have to remove dentry from file handle. The direct result is the file handle for the same file remains unchanged across machine reboot. Rebooting Linux NFS server no longer cause the NFS clients to get the stale NFS file handle.

However, deleting and creating the same pathname may still lead to the same file handle. To solve it, a generation counter is added to the file handle:

- A system wide generation counter is initialized with a random number during machine boot.
- When a new inode is created, it is assigned with the current generation counter and the generation counter is incremented.
- When a freed inode is written to the disk, its generation counter is incremented.

Since we removed the dependency on pathname in the file handle, rename and hard links within the same directory won't cause change in the file handle. However, since the parent directory inode is still the part of the file handle, move a file or hard link a file from one directory to a different directory within the same file system will result in the file handle change. So far we haven't got any report which shows it is a real problem yet.

3.4 Other Considerations

Currently, the 2.2 serious Linux kernel only supports UDP for NFS. Since UDP is supported by all NFS implementations, it shouldn't be a real issue to use Linux as an NFS server in a heterogeneous environment.

The NFS server in the Linux kernel doesn't accept requests from insecure port by default. But some NFS client implementations require to use the insecure ports for NFS. We can disable it in /etc/exports on the host by host basis.

During the tests, we discovered that some NFS client implementations set the size bit for symlink and mkdir. As the result, we changed the Linux NFS server to ignore the size bit for symlink and mkdir.

We also noticed that Solaris NFS automount client made the RPC call to the NULL procedure to verify if the NFS server is available before mounting the exported file system. We changed the Linux NFS server to allow requests to the NFS server NULL procedure from any hosts.

3.5 Results

We performed our tests on the Linux NFS server under Linux 2.2.9 plus kernel patches from knfsd 1.4.2, which contains all the necessary changes to ensure interoperability with other NFS implementations. We ran the Connectathon testsuites on Solaris 7/Sparc client, Solaris 7/x86 client and Linux client.

The Solaris test results are in Appendix A. They showed that the Linux kernel-space NFS server passed all the tests serving Solaris clients. In the Linux test result in Appendix B, there is a warning:

```
Test #13 - Check locking and mmap semantics.
Parent: 13.0  - F_TLOCK [     ffe,  ENDING] PASSED.
Parent: 13.1  - mmap [       0,    1000] WARNING!
Parent: **** Expected EAGAIN, returned success...
```

From the comment in the test:

```
/*
 * Can a file be locked and mapped at same time?  It's
 * potentially safe if the whole file is locked, or if the
 * locked region is page-aligned.  Otherwise, there is a
 * race condition between two clients that lock disjoint
 * regions of the same page.  So for this test we
 * deliberately lock a region that is not
 * page-aligned--that should cause the mmap to fail.  But if
 * it doesn't, that's arguably not an interoperability
 * problem, so make it a warning, not a fatal error.
 */
```

This warning is due to the Linux NFS client implementation. It doesn't affect the interoperability of the Linux NFS server.

4 Performance

We measured the NFS server performances under different server setup for the NFS V2 protocol with UDP. We used SPEC System File Server Suite Release 2.0 (SFS 2.0) from Standard Performance Evaluation Corporation (SPEC). The test was done for both the kernel-space NFS server in Linux kernel 2.2.9 plus the kernel NFS server patches in knfsd 1.4.2 and the user-space NFS server version 2.2beta37. The setup consists of

- A NFS server is a dual 500Mhz Pentium III Xeon 512KB cache with 256MB RAM, 1 Intel 82557 100Mb Ethernet controller, 2 SCSI U-2 NCR53C895 controllers, 1 18GB QUANTUM QM318000TD-SW harddrive on one SCSI controller, 4 9GB SEAGATE ST39102LW harddrives on the other SCSI controller running the Linux kernel 2.2.9 plus knfsd 1.4.2.
- Clients are a 500Mhz Pentium III with 128MB RAM running the Linux kernel 2.2.7, a dual 500Mhz Pentium III with 256MB RAM running the kernel 2.2.9, a quad 400Mhz Pentium III Xeon 1024KB cache with 1GB RAM running the Linux kernel 2.2.7, a Sparc Ultra 10 running Solaris 7 and a 450MHz Pentium II with 64MB RAM running Solaris 7.
- All machines are connected with 100Mb Ethernet switches.

Those performance measurements were only used for the Linux NFS server tuning. They shouldn't be interpreted for any other purposes. To compare the Linux kernel-space NFS server performance with other implementations requires to follow very strict rules setup by SPEC, which is not the intention of this paper.

4.1 The User-Space NFS Server

The user-space NFS server failed the SFS NFS validation test. After skipping the validation test, the benchmark was run. However, it was aborted when the load was over 200. As the result, there were no complete SFS benchmarks for the user-space NFS server. The partial result is

Run	# of Server Threads	# of Clients	# of Processes Per Client	Load (Ops/Sec)	Throughput (Ops/Sec)	Response Time (Msec/Op)
1	1	4	2	100	98	1.8
2	1	4	2	200	200	6.4
3	1	4	2	300	N/A	N/A
4	1	4	2	400	N/A	N/A

4.2 The Kernel-Space NFS Server

We ran the SFS benchmarks with different NFS server configurations under various loads. Before each run, we removed all files and directories on the exported file systems on the NFS server. We didn't run "mke2fs" on them. We only used the "rm -rf" command.

4.2.1 Light Load

We ran the SFS benchmarks with initial load set to 100 for 5 times and incremented the load by 100 at each run. We measured the server performances with different numbers of server threads, 1, 2, 4, 8 and 16. The results were listed here:

Run	# of Server Threads	# of Clients	# of Processes Per Client	Load (Ops/Sec)	Throughput (Ops/Sec)	Response Time (Msec/Op)
1	1	4	2	100	98	1.4
2	1	4	2	200	200	2.7
3	1	4	2	300	302	3.3
4	1	4	2	400	402	6.8
1	2	4	2	100	99	1.0
2	2	4	2	200	200	1.7
3	2	4	2	300	302	2.0
4	2	4	2	400	398	3.3
1	4	4	2	100	98	1.0
2	4	4	2	200	200	1.4
3	4	4	2	300	303	1.5
4	4	4	2	400	398	2.1
1	8	4	2	100	98	1.0
2	8	4	2	200	200	1.4
3	8	4	2	300	302	1.6
4	8	4	2	400	398	2.1

The benchmarks show that even for the light load, the kernel-space NFS server outperforms the user-space one by a wide margin. In this test load, 4 kernel NFS server threads provides adequate performance.

4.2.2 Moderate Load

To simulate the moderate load, we set the initial load to 500 and increment the load by 500 at each run:

Run	# of Server Threads	# of Clients	# of Processes Per Client	Load (Ops/Sec)	Throughput (Ops/Sec)	Response Time (Msec/Op)
1	1	5	2	500	498	8.1
2	1	5	2	1000	380	22.3
3	1	5	2	1500	274	30.9
4	1	5	2	2000	223	38.1
1	2	5	2	500	500	5.5
2	2	5	2	1000	607	14.1
3	2	5	2	1500	453	18.7
4	2	5	2	2000	373	22.8
1	4	5	2	500	499	3.6
2	4	5	2	1000	992	8.1
3	4	5	2	1500	790	11.0
4	4	5	2	2000	612	13.6
1	8	5	2	500	497	2.9
2	8	5	2	1000	1005	4.1
3	8	5	2	1500	1245	9.1
4	8	5	2	2000	939	11.9
1	16	5	2	500	498	3.0
2	16	5	2	1000	1006	4.6
3	16	5	2	1500	1242	9.0
4	16	5	2	2000	924	12.1

The benchmark numbers indicate that under the moderate load, when the number of the server threads is above 8, the bottleneck is the network interface.

4.2.3 Heavy Load

For the heavy load, we set the initial load to 800 and increment the load by 800 at each run:

Run	# of Server Threads	# of Clients	# of Processes Per Client	Load (Ops/Sec)	Throughput (Ops/Sec)	Response Time (Msec/Op)
1	1	5	4	800	356	56.6
2	1	5	4	1600	212	97.8
3	1	5	4	2400	162	127.5
4	1	5	4	3200	138	151.9
1	2	5	4	800	569	36.8
2	2	5	4	1600	381	54.8
3	2	5	4	2400	311	67.5
4	2	5	4	3200	264	78.6
1	4	5	4	800	782	27.4
2	4	5	4	1600	594	35.9
3	4	5	4	2400	494	42.8
4	4	5	4	3200	422	49.8
1	8	5	4	800	799	6.9
2	8	5	4	1600	1125	19.4
3	8	5	4	2400	907	23.9
4	8	5	4	3200	775	27.7
1	16	5	4	800	801	6.1
2	16	5	4	1600	1539	14.2
3	16	5	4	2400	1190	18.9
4	16	5	4	3200	967	22.8

It is quite clear that under the heavy load, the NFS server performance benefits tremendously when the number of the server threads was increased from 1 to 8. When it is above 8, the performance gain was not as significant as the lower thread count.

Overall, the SFS benchmarks show that

- The kernel-space NFS server is a major step forward in the Linux NFS server implementation. Its interoperability and performance are way beyond the old NFS server running in user-space.
- The number of kernel NFS server threads will affect the server performance. In general, the more, the better.
- Under the light load, the performance gain from increasing the number of server threads is not very significant.
- Under the moderate load, 8 server threads seem to work best for our server hardware configuration.
- Under the heavy load, the performance gain from increasing the number of server threads is significant even at 16. However, 16 server threads take significant kernel sources.

For our specific server hardware setup, 8 kernel NFS server threads strike a balance between performance and resource usage.

To further study the Linux NFS server performance, we need to run the benchmarks on

- Different number of CPUs.
- Different number of SCSI controllers.
- Different number of harddrives.
- RAID storage.
- More RAM.
- Gigabit Ethernet.

4.2.4 Other Factors

The NFS server related programs depend on hostname lookup for correct operation. glibc 2.1.1 or above is recommended for better hostname lookup performance. If NIS/NIS+ is used, the Name Switch Cache Daemon (nscd) in glibc should be started to further improve hostname lookup speed.

5 Future Development

Today's kernel NFS server is very much usable, but it not perfect. People are working on further improvements.

- The current stable Linux kernel only supports NFS over UDP. We like to get NFS over TCP stabilized.
- We are also working on support for the NFS Version 3 protocol.
- As we mentioned above, rename or move a file to a different directory on the same file system still causes the change in the file handle. We'd like to address it in the future development.

  One proposal looks very promising. That is make the mapping between file and file handle be part of the VFS layer. Each file system will provide a conversion between a file and a file handle, which is persistent between server reboots as well as independent of its pathnames as well as its parent directory pathnames. It also solves the problem where the underlying file systems do not support all the fields in the current file handle format.

6 Conclusion

The NFS server implementation of the NFS V2 protocol with UDP protocol in the 2.2 series Linux kernel plus the kernel patches in knfsd 1.4.2 works with Solaris and Linux NFS clients. With the multi-threaded kernel NFS server improves the NFS server performance dramatically. It shows that Linux on the adequate hardwares makes a viable NFS server option for enterprise.

References

[1] Network Working Group, Sun Microsystems, Inc. *NFS: Network File System Protocol Specification*, Request for Comments: 1094, March 1989

A Solaris Clients with Linux Kernel-Space NFS server

Here are the Connectathon testsuites on Solaris 7/Sparc client and Solaris 7/x86 client.

A.1 Solaris 7/Sparc client

```
Start tests on path /mnt/nfstest/fudan.test [y/n]?
sh ./runtests -a -t /mnt/nfstest/fudan.test

Starting BASIC tests: test directory /mnt/nfstest/fudan.test (arg: -t)

test1: File and directory creation test
created 155 files 62 directories 5 levels deep in 0.31 seconds
test1 ok.

test2: File and directory removal test
removed 155 files 62 directories 5 levels deep in 0.45 seconds
test2 ok.

test3: lookups across mount point
500 getcwd and stat calls in 0.1  seconds
test3 ok.

test4: setattr, getattr, and lookup
1000 chmods and stats on 10 files in 0.34 seconds
test4 ok.

test5: read and write
wrote 1048576 byte file 10 times in 0.94 seconds (11135895 bytes/sec)
read 1048576 byte file 10 times in 0.91 seconds (11477357 bytes/sec)
test5 ok.

test6: readdir
20500 entries read, 200 files in 0.83 seconds
test6 ok.

test7: link and rename
200 renames and links on 10 files in 0.57 seconds
```

```
test7 ok.

test8: symlink and readlink
400 symlinks and readlinks on 10 files in 0.65 seconds
test8 ok.

test9: statvfs
1500 statvfs calls in 0.44 seconds
test9 ok.

Congratulations, you passed the basic tests!

GENERAL TESTS: directory /mnt/nfstest/fudan.test
cd /mnt/nfstest/fudan.test; rm -f Makefile runtests runtests.wrk *.sh *.c mkdummy rmdummy  nroff.i
cp Makefile runtests runtests.wrk *.sh *.c mkdummy rmdummy  nroff.in makefile.tst /mnt/nfstest/fuda

Small Compile
0.4 (0.0) real 0.1 (0.1) user 0.0 (0.0) sys

Tbl
0.0 (0.0) real 0.0 (0.0) user 0.0 (0.0) sys

Nroff
0.0 (0.0) real 0.0 (0.0) user 0.0 (0.0) sys

Large Compile
0.6 (0.0) real 0.3 (0.0) user 0.0 (0.0) sys

Four simultaneous large compiles
2.0 (0.1) real 1.3 (0.1) user 0.4 (0.0) sys

Makefile
0.2 (0.0) real 0.1 (0.0) user 0.1 (0.0) sys

General tests complete

SPECIAL TESTS: directory /mnt/nfstest/fudan.test
cd /mnt/nfstest/fudan.test; rm -f runtests runtests.wrk READWIN.txt Makefile op_unlk op_ren op_chmo
cp runtests runtests.wrk READWIN.txt Makefile op_unlk op_ren op_chmod dupreq excltest negseek renam

check for proper open/unlink operation
nfsjunk files before unlink:
  ./nfsAAAbCayvn open; unlink ret = 0
nfsjunk files after unlink:
  -rw-rw-rw-   1 spec     spec           0 Jun 21 16:03 .nfs6884
```

```
data compare ok
nfsjunk files after close:
  -rw-rw-rw-   1 spec     spec         100 Jun 21 16:03 .nfs6884
test completed successfully.

check for proper open/rename operation
nfsjunk files before rename:
  -rw-rw-rw-   1 spec     spec         100 Jun 21 16:03 .nfs6884
./nfsbBAAKDa4vn open; rename ret = 0
nfsjunk files after rename:
  -rw-rw-rw-   1 spec     spec         100 Jun 21 16:03 .nfs6884
-rwxrwxrwx   1 spec     spec           0 Jun 21 16:03 .nfs7884
data compare ok
nfsjunk files after close:
  -rw-rw-rw-   1 spec     spec         100 Jun 21 16:03 .nfs6884
-rwxrwxrwx   1 spec     spec         100 Jun 21 16:03 .nfs7884
test completed successfully.

check for proper open/chmod 0 operation
testfile before chmod:
  -rw-rw-rw-   1 spec     spec           0 Jun 21 16:03 ./nfsAAAhFaywn
./nfsAAAhFaywn open; chmod ret = 0
testfile after chmod:
  ----------   1 spec     spec           0 Jun 21 16:03 ./nfsAAAhFaywn
data compare ok
testfile after write/read:
  ----------   1 spec     spec         100 Jun 21 16:03 ./nfsAAAhFaywn
test completed successfully.

check for lost reply on non-idempotent requests
100 tries

test exclusive create.

test negative seek, you should get: read: Invalid argument
or lseek: Invalid argument

test rename

test truncate
truncate succeeded

test holey file support
Holey file test ok
```

```
second check for lost reply on non-idempotent requests
testing 50 idempotencies in directory "testdir"

test rewind support

test telldir cookies (slow)

write/read 30 MB file


test freesp and file size

Special tests complete

Starting LOCKING tests: test directory /mnt/nfstest/fudan.test (arg: -t)

Testing native post-LFS locking

Creating parent/child synchronization pipes.

Test #1 - Test regions of an unlocked file.
Parent: 1.1  - F_TEST  [       0,       1] PASSED.
Parent: 1.2  - F_TEST  [       0,  ENDING] PASSED.
Parent: 1.3  - F_TEST  [       0,7fffffff] PASSED.
Parent: 1.4  - F_TEST  [       1,       1] PASSED.
Parent: 1.5  - F_TEST  [       1,  ENDING] PASSED.
Parent: 1.6  - F_TEST  [       1,7fffffff] PASSED.
Parent: 1.7  - F_TEST  [7fffffff,       1] PASSED.
Parent: 1.8  - F_TEST  [7fffffff,  ENDING] PASSED.
Parent: 1.9  - F_TEST  [7fffffff,7fffffff] PASSED.

Test #2 - Try to lock the whole file.
Parent: 2.0  - F_TLOCK [       0,  ENDING] PASSED.
Child:  2.1  - F_TEST  [       0,       1] PASSED.
Child:  2.2  - F_TEST  [       0,  ENDING] PASSED.
Child:  2.3  - F_TEST  [       0,7fffffff] PASSED.
Child:  2.4  - F_TEST  [       1,       1] PASSED.
Child:  2.5  - F_TEST  [       1,  ENDING] PASSED.
Child:  2.6  - F_TEST  [       1,7fffffff] PASSED.
Child:  2.7  - F_TEST  [7fffffff,       1] PASSED.
Child:  2.8  - F_TEST  [7fffffff,  ENDING] PASSED.
Child:  2.9  - F_TEST  [7fffffff,7fffffff] PASSED.
Parent: 2.10 - F_ULOCK [       0,  ENDING] PASSED.

Test #3 - Try to lock just the 1st byte.
```

```
Parent: 3.0  - F_TLOCK [       0,       1] PASSED.
Child:  3.1  - F_TEST  [       0,       1] PASSED.
Child:  3.2  - F_TEST  [       0,  ENDING] PASSED.
Child:  3.3  - F_TEST  [       1,       1] PASSED.
Child:  3.4  - F_TEST  [       1,  ENDING] PASSED.
Parent: 3.5  - F_ULOCK [       0,       1] PASSED.

Test #4 - Try to lock the 2nd byte, test around it.
Parent: 4.0  - F_TLOCK [       1,       1] PASSED.
Child:  4.1  - F_TEST  [       0,       1] PASSED.
Child:  4.2  - F_TEST  [       0,       2] PASSED.
Child:  4.3  - F_TEST  [       0,  ENDING] PASSED.
Child:  4.4  - F_TEST  [       1,       1] PASSED.
Child:  4.5  - F_TEST  [       1,       2] PASSED.
Child:  4.6  - F_TEST  [       1,  ENDING] PASSED.
Child:  4.7  - F_TEST  [       2,       1] PASSED.
Child:  4.8  - F_TEST  [       2,       2] PASSED.
Child:  4.9  - F_TEST  [       2,  ENDING] PASSED.
Parent: 4.10 - F_ULOCK [       1,       1] PASSED.

Test #5 - Try to lock 1st and 2nd bytes, test around them.
Parent: 5.0  - F_TLOCK [       0,       1] PASSED.
Parent: 5.1  - F_TLOCK [       2,       1] PASSED.
Child:  5.2  - F_TEST  [       0,       1] PASSED.
Child:  5.3  - F_TEST  [       0,       2] PASSED.
Child:  5.4  - F_TEST  [       0,  ENDING] PASSED.
Child:  5.5  - F_TEST  [       1,       1] PASSED.
Child:  5.6  - F_TEST  [       1,       2] PASSED.
Child:  5.7  - F_TEST  [       1,  ENDING] PASSED.
Child:  5.8  - F_TEST  [       2,       1] PASSED.
Child:  5.9  - F_TEST  [       2,       2] PASSED.
Child:  5.10 - F_TEST  [       2,  ENDING] PASSED.
Child:  5.11 - F_TEST  [       3,       1] PASSED.
Child:  5.12 - F_TEST  [       3,       2] PASSED.
Child:  5.13 - F_TEST  [       3,  ENDING] PASSED.
Parent: 5.14 - F_ULOCK [       0,       1] PASSED.
Parent: 5.15 - F_ULOCK [       2,       1] PASSED.

Test #6 - Try to lock the MAXEOF byte.
Parent: 6.0  - F_TLOCK [7fffffff,       1] PASSED.
Child:  6.1  - F_TEST  [7ffffffe,       1] PASSED.
Child:  6.2  - F_TEST  [7ffffffe,       2] PASSED.
Child:  6.3  - F_TEST  [7ffffffe,  ENDING] PASSED.
Child:  6.4  - F_TEST  [7fffffff,       1] PASSED.
Child:  6.5  - F_TEST  [7fffffff,       2] PASSED.
```

```
Child:  6.6  - F_TEST  [7fffffff,  ENDING] PASSED.
Child:  6.7  - F_TEST  [80000000,  ENDING] PASSED.
Child:  6.8  - F_TEST  [80000000,       1] PASSED.
Child:  6.9  - F_TEST  [80000000,7fffffff] PASSED.
Child:  6.10 - F_TEST  [80000000,80000000] PASSED.
Parent: 6.11 - F_ULOCK [7fffffff,       1] PASSED.

Test #7 - Test parent/child mutual exclusion.
Parent: 7.0  - F_TLOCK [       0,       4] PASSED.
Parent: Wrote ’aaaa’ to testfile [ 0, 4 ].
Parent: Now free child to run, should block on lock.
Parent: Check data in file to insure child blocked.
Parent: Read ’aaaa’ from testfile [ 0, 4 ].
Parent: 7.1  - COMPARE [       0,       4] PASSED.
Parent: Now unlock region so child will unblock.
Parent: 7.2  - F_ULOCK [       0,       4] PASSED.
Child:  7.3  - F_LOCK  [       0,       4] PASSED.
Child:  Write child’s version of the data and release lock.
Parent: Now try to regain lock, parent should block.
Child:  Wrote ’bbbb’ to testfile [ 0, 4 ].
Child:  7.4  - F_ULOCK [       0,       4] PASSED.
Parent: 7.5  - F_LOCK  [       0,       4] PASSED.
Parent: Check data in file to insure child unblocked.
Parent: Read ’bbbb’ from testfile [ 0, 4 ].
Parent: 7.6  - COMPARE [       0,       4] PASSED.
Parent: 7.7  - F_ULOCK [       0,       4] PASSED.

Test #8 - Rate test performing lock/unlock cycles.
Parent: Performed 1000 lock/unlock cycles in 1020 msecs. [117647 lpm].

Test #10 - Make sure a locked region is split properly.
Parent: 10.0  - F_TLOCK [       0,       3] PASSED.
Parent: 10.1  - F_ULOCK [       1,       1] PASSED.
Child:  10.2  - F_TEST  [       0,       1] PASSED.
Child:  10.3  - F_TEST  [       2,       1] PASSED.
Child:  10.4  - F_TEST  [       3,  ENDING] PASSED.
Child:  10.5  - F_TEST  [       1,       1] PASSED.
Parent: 10.6  - F_ULOCK [       0,       1] PASSED.
Parent: 10.7  - F_ULOCK [       2,       1] PASSED.
Child:  10.8  - F_TEST  [       0,       3] PASSED.
Parent: 10.9  - F_ULOCK [       0,       1] PASSED.
Parent: 10.10 - F_TLOCK [       1,       3] PASSED.
Parent: 10.11 - F_ULOCK [       2,       1] PASSED.
Child:  10.12 - F_TEST  [       1,       1] PASSED.
Child:  10.13 - F_TEST  [       3,       1] PASSED.
```

```
Child:  10.14 - F_TEST  [       4,  ENDING] PASSED.
Child:  10.15 - F_TEST  [       2,       1] PASSED.
Child:  10.16 - F_TEST  [       0,       1] PASSED.

Test #11 - Make sure close() releases the process's locks.
Parent: 11.0  - F_TLOCK [       0,  ENDING] PASSED.
Parent: Closed testfile.
Child:  11.1  - F_TLOCK [       0,  ENDING] PASSED.
Child:  11.2  - F_ULOCK [       0,  ENDING] PASSED.
Parent: 11.3  - F_TLOCK [      1d,     5b7] PASSED.
Parent: 11.4  - F_TLOCK [    2000,      57] PASSED.
Parent: Closed testfile.
Child:  11.5  - F_TLOCK [       0,  ENDING] PASSED.
Child:  11.6  - F_ULOCK [       0,  ENDING] PASSED.
Parent: Wrote '123456789abcdef' to testfile [ 0, 16 ].
Parent: 11.7  - F_TLOCK [       0,  ENDING] PASSED.
Parent: Wrote '123456789abcdef' to testfile [ 13, 16 ].
Parent: Closed testfile.
Child:  11.8  - F_TLOCK [       0,  ENDING] PASSED.
Child:  11.9  - F_ULOCK [       0,  ENDING] PASSED.
Parent: Wrote '123456789abcdef' to testfile [ 0, 16 ].
Parent: 11.10 - F_TLOCK [       0,  ENDING] PASSED.
Parent: Truncated testfile.
Parent: Closed testfile.
Child:  11.11 - F_TLOCK [       0,  ENDING] PASSED.
Child:  11.12 - F_ULOCK [       0,  ENDING] PASSED.

Test #12 - Signalled process should release locks.
Child:  12.0  - F_TLOCK [       0,  ENDING] PASSED.
Parent: Killed child process.
Parent: 12.1  - F_TLOCK [       0,  ENDING] PASSED.

Test #13 - Check locking and mmap semantics.
Parent: 13.0  - F_TLOCK [    1ffe,  ENDING] PASSED.
Parent: 13.1  - mmap [       0,    2000] PASSED.
Parent: 13.2  - F_ULOCK [       0,  ENDING] PASSED.
Parent: 13.3  - mmap [       0,    2000] PASSED.
Parent: 13.4  - F_TLOCK [    1ffe,  ENDING] PASSED.

** PARENT pass 1 results: 45/45 pass, 0/0 warn, 0/0 fail (pass/total).

**  CHILD pass 1 results: 64/64 pass, 0/0 warn, 0/0 fail (pass/total).
Congratulations, you passed the locking tests!

All tests completed
```

A.2 Solaris 7/x86 client

```
Start tests on path /mnt/nfstest/beijingp.test [y/n]?
sh ./runtests -a -t /mnt/nfstest/beijingp.test

Starting BASIC tests: test directory /mnt/nfstest/beijingp.test (arg: -t)

test1: File and directory creation test
created 155 files 62 directories 5 levels deep in 0.29 seconds
test1 ok.

test2: File and directory removal test
removed 155 files 62 directories 5 levels deep in 0.34 seconds
test2 ok.

test3: lookups across mount point
500 getcwd and stat calls in 0.1  seconds
test3 ok.

test4: setattr, getattr, and lookup
1000 chmods and stats on 10 files in 0.33 seconds
test4 ok.

test5: read and write
wrote 1048576 byte file 10 times in 0.94 seconds (11144808 bytes/sec)
read 1048576 byte file 10 times in 7.96 seconds (1316127 bytes/sec)
test5 ok.

test6: readdir
20500 entries read, 200 files in 1.21 seconds
test6 ok.

test7: link and rename
200 renames and links on 10 files in 0.90 seconds
test7 ok.

test8: symlink and readlink
400 symlinks and readlinks on 10 files in 0.90 seconds
test8 ok.

test9: statvfs
1500 statvfs calls in 0.66 seconds
test9 ok.

Congratulations, you passed the basic tests!
```

```
GENERAL TESTS: directory /mnt/nfstest/beijingp.test
cd /mnt/nfstest/beijingp.test; rm -f Makefile runtests runtests.wrk *.sh *.c mkdummy rmdummy  nroff
cp Makefile runtests runtests.wrk *.sh *.c mkdummy rmdummy  nroff.in makefile.tst /mnt/nfstest/beij

Small Compile
0.5 (0.5) real 0.1 (0.0) user 0.0 (0.0) sys

Tbl
0.0 (0.0) real 0.0 (0.0) user 0.0 (0.0) sys

Nroff
0.0 (0.0) real 0.0 (0.0) user 0.0 (0.0) sys

Large Compile
0.5 (0.1) real 0.2 (0.0) user 0.0 (0.1) sys

Four simultaneous large compiles
1.6 (0.3) real 0.9 (0.1) user 0.3 (0.1) sys

Makefile
0.3 (0.0) real 0.1 (0.0) user 0.2 (0.1) sys

General tests complete

SPECIAL TESTS: directory /mnt/nfstest/beijingp.test
cd /mnt/nfstest/beijingp.test; rm -f runtests runtests.wrk READWIN.txt Makefile op_unlk op_ren op_c
cp runtests runtests.wrk READWIN.txt Makefile op_unlk op_ren op_chmod dupreq excltest negseek renam

check for proper open/unlink operation
nfsjunk files before unlink:
  ./nfsAAAu.a0Ah open; unlink ret = 0
nfsjunk files after unlink:
  -rw-rw-rw-   1 spec     spec           0 Jun 21 16:04 .nfs702C
data compare ok
nfsjunk files after close:
  -rw-rw-rw-   1 spec     spec         100 Jun 21 16:04 .nfs702C
test completed successfully.

check for proper open/rename operation
nfsjunk files before rename:
  -rw-rw-rw-   1 spec     spec         100 Jun 21 16:04 .nfs702C
./nfsbBAAY_aiBh open; rename ret = 0
nfsjunk files after rename:
  -rw-rw-rw-   1 spec     spec         100 Jun 21 16:04 .nfs702C
```

```
-rwxrwxrwx   1 spec     spec           0 Jun 21 16:04 .nfs802C
data compare ok
nfsjunk files after close:
  -rw-rw-rw-   1 spec     spec         100 Jun 21 16:04 .nfs702C
-rwxrwxrwx   1 spec     spec         100 Jun 21 16:04 .nfs802C
test completed successfully.

check for proper open/chmod 0 operation
testfile before chmod:
  -rw-rw-rw-   1 spec     spec           0 Jun 21 16:04 ./nfsAAAhbaOBh
./nfsAAAhbaOBh open; chmod ret = 0
testfile after chmod:
  ----------   1 spec     spec           0 Jun 21 16:04 ./nfsAAAhbaOBh
data compare ok
testfile after write/read:
  ----------   1 spec     spec         100 Jun 21 16:04 ./nfsAAAhbaOBh
test completed successfully.

check for lost reply on non-idempotent requests
100 tries

test exclusive create.

test negative seek, you should get: read: Invalid argument
or lseek: Invalid argument

test rename

test truncate
truncate succeeded

test holey file support
Holey file test ok

second check for lost reply on non-idempotent requests
testing 50 idempotencies in directory "testdir"

test rewind support

test telldir cookies (slow)

write/read 30 MB file


test freesp and file size
```

```
Special tests complete

Starting LOCKING tests: test directory /mnt/nfstest/beijingp.test (arg: -t)

Testing native post-LFS locking

Creating parent/child synchronization pipes.

Test #1 - Test regions of an unlocked file.
Parent: 1.1  - F_TEST  [       0,       1] PASSED.
Parent: 1.2  - F_TEST  [       0,  ENDING] PASSED.
Parent: 1.3  - F_TEST  [       0,7fffffff] PASSED.
Parent: 1.4  - F_TEST  [       1,       1] PASSED.
Parent: 1.5  - F_TEST  [       1,  ENDING] PASSED.
Parent: 1.6  - F_TEST  [       1,7fffffff] PASSED.
Parent: 1.7  - F_TEST  [7fffffff,       1] PASSED.
Parent: 1.8  - F_TEST  [7fffffff,  ENDING] PASSED.
Parent: 1.9  - F_TEST  [7fffffff,7fffffff] PASSED.

Test #2 - Try to lock the whole file.
Parent: 2.0  - F_TLOCK [       0,  ENDING] PASSED.
Child:  2.1  - F_TEST  [       0,       1] PASSED.
Child:  2.2  - F_TEST  [       0,  ENDING] PASSED.
Child:  2.3  - F_TEST  [       0,7fffffff] PASSED.
Child:  2.4  - F_TEST  [       1,       1] PASSED.
Child:  2.5  - F_TEST  [       1,  ENDING] PASSED.
Child:  2.6  - F_TEST  [       1,7fffffff] PASSED.
Child:  2.7  - F_TEST  [7fffffff,       1] PASSED.
Child:  2.8  - F_TEST  [7fffffff,  ENDING] PASSED.
Child:  2.9  - F_TEST  [7fffffff,7fffffff] PASSED.
Parent: 2.10 - F_ULOCK [       0,  ENDING] PASSED.

Test #3 - Try to lock just the 1st byte.
Parent: 3.0  - F_TLOCK [       0,       1] PASSED.
Child:  3.1  - F_TEST  [       0,       1] PASSED.
Child:  3.2  - F_TEST  [       0,  ENDING] PASSED.
Child:  3.3  - F_TEST  [       1,       1] PASSED.
Child:  3.4  - F_TEST  [       1,  ENDING] PASSED.
Parent: 3.5  - F_ULOCK [       0,       1] PASSED.

Test #4 - Try to lock the 2nd byte, test around it.
Parent: 4.0  - F_TLOCK [       1,       1] PASSED.
Child:  4.1  - F_TEST  [       0,       1] PASSED.
Child:  4.2  - F_TEST  [       0,       2] PASSED.
```

```
Child:  4.3  - F_TEST  [       0,  ENDING] PASSED.
Child:  4.4  - F_TEST  [       1,       1] PASSED.
Child:  4.5  - F_TEST  [       1,       2] PASSED.
Child:  4.6  - F_TEST  [       1,  ENDING] PASSED.
Child:  4.7  - F_TEST  [       2,       1] PASSED.
Child:  4.8  - F_TEST  [       2,       2] PASSED.
Child:  4.9  - F_TEST  [       2,  ENDING] PASSED.
Parent: 4.10 - F_ULOCK [       1,       1] PASSED.

Test #5 - Try to lock 1st and 2nd bytes, test around them.
Parent: 5.0  - F_TLOCK [       0,       1] PASSED.
Parent: 5.1  - F_TLOCK [       2,       1] PASSED.
Child:  5.2  - F_TEST  [       0,       1] PASSED.
Child:  5.3  - F_TEST  [       0,       2] PASSED.
Child:  5.4  - F_TEST  [       0,  ENDING] PASSED.
Child:  5.5  - F_TEST  [       1,       1] PASSED.
Child:  5.6  - F_TEST  [       1,       2] PASSED.
Child:  5.7  - F_TEST  [       1,  ENDING] PASSED.
Child:  5.8  - F_TEST  [       2,       1] PASSED.
Child:  5.9  - F_TEST  [       2,       2] PASSED.
Child:  5.10 - F_TEST  [       2,  ENDING] PASSED.
Child:  5.11 - F_TEST  [       3,       1] PASSED.
Child:  5.12 - F_TEST  [       3,       2] PASSED.
Child:  5.13 - F_TEST  [       3,  ENDING] PASSED.
Parent: 5.14 - F_ULOCK [       0,       1] PASSED.
Parent: 5.15 - F_ULOCK [       2,       1] PASSED.

Test #6 - Try to lock the MAXEOF byte.
Parent: 6.0  - F_TLOCK [7fffffff,       1] PASSED.
Child:  6.1  - F_TEST  [7ffffffe,       1] PASSED.
Child:  6.2  - F_TEST  [7ffffffe,       2] PASSED.
Child:  6.3  - F_TEST  [7ffffffe,  ENDING] PASSED.
Child:  6.4  - F_TEST  [7fffffff,       1] PASSED.
Child:  6.5  - F_TEST  [7fffffff,       2] PASSED.
Child:  6.6  - F_TEST  [7fffffff,  ENDING] PASSED.
Child:  6.7  - F_TEST  [80000000,  ENDING] PASSED.
Child:  6.8  - F_TEST  [80000000,       1] PASSED.
Child:  6.9  - F_TEST  [80000000,7fffffff] PASSED.
Child:  6.10 - F_TEST  [80000000,80000000] PASSED.
Parent: 6.11 - F_ULOCK [7fffffff,       1] PASSED.

Test #7 - Test parent/child mutual exclusion.
Parent: 7.0  - F_TLOCK [       0,       4] PASSED.
Parent: Wrote 'aaaa' to testfile [ 0, 4 ].
Parent: Now free child to run, should block on lock.
```

```
Parent: Check data in file to insure child blocked.
Parent: Read 'aaaa' from testfile [ 0, 4 ].
Parent: 7.1  - COMPARE [       0,       4] PASSED.
Parent: Now unlock region so child will unblock.
Parent: 7.2  - F_ULOCK [       0,       4] PASSED.
Child:  7.3  - F_LOCK  [       0,       4] PASSED.
Child:  Write child's version of the data and release lock.
Parent: Now try to regain lock, parent should block.
Child:  Wrote 'bbbb' to testfile [ 0, 4 ].
Parent: 7.5  - F_LOCK  [       0,       4] PASSED.
Parent: Check data in file to insure child unblocked.
Child:  7.4  - F_ULOCK [       0,       4] PASSED.
Parent: Read 'bbbb' from testfile [ 0, 4 ].
Parent: 7.6  - COMPARE [       0,       4] PASSED.
Parent: 7.7  - F_ULOCK [       0,       4] PASSED.

Test #8 - Rate test performing lock/unlock cycles.
Parent: Performed 1000 lock/unlock cycles in 720 msecs. [166666 lpm].

Test #10 - Make sure a locked region is split properly.
Parent: 10.0  - F_TLOCK [       0,       3] PASSED.
Parent: 10.1  - F_ULOCK [       1,       1] PASSED.
Child:  10.2  - F_TEST  [       0,       1] PASSED.
Child:  10.3  - F_TEST  [       2,       1] PASSED.
Child:  10.4  - F_TEST  [       3,  ENDING] PASSED.
Child:  10.5  - F_TEST  [       1,       1] PASSED.
Parent: 10.6  - F_ULOCK [       0,       1] PASSED.
Parent: 10.7  - F_ULOCK [       2,       1] PASSED.
Child:  10.8  - F_TEST  [       0,       3] PASSED.
Parent: 10.9  - F_ULOCK [       0,       1] PASSED.
Parent: 10.10 - F_TLOCK [       1,       3] PASSED.
Parent: 10.11 - F_ULOCK [       2,       1] PASSED.
Child:  10.12 - F_TEST  [       1,       1] PASSED.
Child:  10.13 - F_TEST  [       3,       1] PASSED.
Child:  10.14 - F_TEST  [       4,  ENDING] PASSED.
Child:  10.15 - F_TEST  [       2,       1] PASSED.
Child:  10.16 - F_TEST  [       0,       1] PASSED.

Test #11 - Make sure close() releases the process's locks.
Parent: 11.0  - F_TLOCK [       0,  ENDING] PASSED.
Parent: Closed testfile.
Child:  11.1  - F_TLOCK [       0,  ENDING] PASSED.
Child:  11.2  - F_ULOCK [       0,  ENDING] PASSED.
Parent: 11.3  - F_TLOCK [      1d,     5b7] PASSED.
Parent: 11.4  - F_TLOCK [    2000,      57] PASSED.
```

```
Parent: Closed testfile.
Child:  11.5  - F_TLOCK [       0,  ENDING] PASSED.
Child:  11.6  - F_ULOCK [       0,  ENDING] PASSED.
Parent: Wrote '123456789abcdef' to testfile [ 0, 16 ].
Parent: 11.7  - F_TLOCK [       0,  ENDING] PASSED.
Parent: Wrote '123456789abcdef' to testfile [ 13, 16 ].
Parent: Closed testfile.
Child:  11.8  - F_TLOCK [       0,  ENDING] PASSED.
Child:  11.9  - F_ULOCK [       0,  ENDING] PASSED.
Parent: Wrote '123456789abcdef' to testfile [ 0, 16 ].
Parent: 11.10 - F_TLOCK [       0,  ENDING] PASSED.
Parent: Truncated testfile.
Parent: Closed testfile.
Child:  11.11 - F_TLOCK [       0,  ENDING] PASSED.
Child:  11.12 - F_ULOCK [       0,  ENDING] PASSED.

Test #12 - Signalled process should release locks.
Child:  12.0  - F_TLOCK [       0,  ENDING] PASSED.
Parent: Killed child process.
Parent: 12.1  - F_TLOCK [       0,  ENDING] PASSED.

Test #13 - Check locking and mmap semantics.
Parent: 13.0  - F_TLOCK [     ffe,  ENDING] PASSED.
Parent: 13.1  - mmap [       0,    1000] PASSED.
Parent: 13.2  - F_ULOCK [       0,  ENDING] PASSED.
Parent: 13.3  - mmap [       0,    1000] PASSED.
Parent: 13.4  - F_TLOCK [     ffe,  ENDING] PASSED.

** PARENT pass 1 results: 45/45 pass, 0/0 warn, 0/0 fail (pass/total).

**  CHILD pass 1 results: 64/64 pass, 0/0 warn, 0/0 fail (pass/total).
Congratulations, you passed the locking tests!

All tests completed
```

B Linux Client with Linux Kernel-Space NFS server

Here is the Connectathon testsuites on Linux client.

```
Start tests on path /mnt/nfstest/osmium.test [y/n]?
sh ./runtests -a -t /mnt/nfstest/osmium.test

Starting BASIC tests: test directory /mnt/nfstest/osmium.test (arg: -t)
```

```
test1: File and directory creation test
created 155 files 62 directories 5 levels deep in 1.6  seconds
test1 ok.

test2: File and directory removal test
removed 155 files 62 directories 5 levels deep in 0.24 seconds
test2 ok.

test3: lookups across mount point
500 getcwd and stat calls in 0.0  seconds
test3 ok.

test4: setattr, getattr, and lookup
1000 chmods and stats on 10 files in 1.13 seconds
test4 ok.

test5: read and write
wrote 1048576 byte file 10 times in 1.95 seconds (5365632 bytes/sec)
read 1048576 byte file 10 times in 0.3  seconds (281572502 bytes/sec)
test5 ok.

test6: readdir
20500 entries read, 200 files in 0.59 seconds
test6 ok.

test7: link and rename
200 renames and links on 10 files in 0.33 seconds
test7 ok.

test8: symlink and readlink
400 symlinks and readlinks on 10 files in 0.43 seconds
test8 ok.

test9: statfs
1500 statfs calls in 0.65 seconds
test9 ok.

Congratulations, you passed the basic tests!

GENERAL TESTS: directory /mnt/nfstest/osmium.test
cd /mnt/nfstest/osmium.test; rm -f Makefile runtests runtests.wrk *.sh *.c mkdummy rmdummy nroff.i
cp Makefile runtests runtests.wrk *.sh *.c mkdummy rmdummy nroff.in makefile.tst /mnt/nfstest/osmiu

Small Compile
  File: "smcomp.time"
```

```
  Size: 661          Filetype: Regular File
  Mode: (0666/-rw-rw-rw-)         Uid: (10000/    spec)  Gid: (10000/    spec)
Device:  0,12  Inode: 12303     Links: 1
Access: Mon Jun 21 16:03:57 1999(00000.00:00:01)
Modify: Mon Jun 21 16:03:58 1999(00000.00:00:00)
Change: Mon Jun 21 16:03:58 1999(00000.00:00:00)

Tbl
  File: "tbl.time"
  Size: 1625         Filetype: Regular File
  Mode: (0666/-rw-rw-rw-)         Uid: (10000/    spec)  Gid: (10000/    spec)
Device:  0,12  Inode: 12306     Links: 1
Access: Mon Jun 21 16:03:58 1999(00000.00:00:00)
Modify: Mon Jun 21 16:03:58 1999(00000.00:00:00)
Change: Mon Jun 21 16:03:58 1999(00000.00:00:00)

Nroff
  File: "nroff.time"
  Size: 654          Filetype: Regular File
  Mode: (0666/-rw-rw-rw-)         Uid: (10000/    spec)  Gid: (10000/    spec)
Device:  0,12  Inode: 12308     Links: 1
Access: Mon Jun 21 16:03:58 1999(00000.00:00:00)
Modify: Mon Jun 21 16:03:58 1999(00000.00:00:00)
Change: Mon Jun 21 16:03:58 1999(00000.00:00:00)

Large Compile
  File: "lrgcomp.time"
  Size: 1595         Filetype: Regular File
  Mode: (0666/-rw-rw-rw-)         Uid: (10000/    spec)  Gid: (10000/    spec)
Device:  0,12  Inode: 12305     Links: 1
Access: Mon Jun 21 16:03:58 1999(00000.00:00:02)
Modify: Mon Jun 21 16:04:00 1999(00000.00:00:00)
Change: Mon Jun 21 16:04:00 1999(00000.00:00:00)

Four simultaneous large compiles
  File: "4lrg.time"
  Size: 4460         Filetype: Regular File
  Mode: (0666/-rw-rw-rw-)         Uid: (10000/    spec)  Gid: (10000/    spec)
Device:  0,12  Inode: 12307     Links: 1
Access: Mon Jun 21 16:04:00 1999(00000.00:00:02)
Modify: Mon Jun 21 16:04:02 1999(00000.00:00:00)
Change: Mon Jun 21 16:04:02 1999(00000.00:00:00)

Makefile
  File: "makefile.time"
```

```
  Size: 645          Filetype: Regular File
  Mode: (0666/-rw-rw-rw-)         Uid: (10000/    spec)  Gid: (10000/    spec)
Device:  0,12  Inode: 12359     Links: 1
Access: Mon Jun 21 16:04:02 1999(00000.00:00:02)
Modify: Mon Jun 21 16:04:03 1999(00000.00:00:01)
Change: Mon Jun 21 16:04:03 1999(00000.00:00:01)

General tests complete

SPECIAL TESTS: directory /mnt/nfstest/osmium.test
cd /mnt/nfstest/osmium.test; rm -f runtests runtests.wrk READWIN.txt Makefile op_unlk op_ren op_chm
cp runtests runtests.wrk READWIN.txt Makefile op_unlk op_ren op_chmod dupreq excltest negseek renam

check for proper open/unlink operation
nfsjunk files before unlink:
  ./nfsfMUDri open; unlink ret = 0
nfsjunk files after unlink:
  -rw-rw-rw-   1 spec     spec            0 Jun 21 16:04 .nfs000030190000000f
data compare ok
nfsjunk files after close:
  test completed successfully.

check for proper open/rename operation
nfsjunk files before rename:
  ./nfsbebYkBv open; rename ret = 0
nfsjunk files after rename:
  -rwxrwxrwx   1 spec     spec            0 Jun 21 16:04 .nfs0000301a00000010
data compare ok
nfsjunk files after close:
  test completed successfully.

check for proper open/chmod 0 operation
testfile before chmod:
  -rw-rw-rw-   1 spec     spec            0 Jun 21 16:04 ./nfslxPegn
./nfslxPegn open; chmod ret = 0
testfile after chmod:
  ----------   1 spec     spec            0 Jun 21 16:04 ./nfslxPegn
data compare ok
testfile after write/read:
  ----------   1 spec     spec          100 Jun 21 16:04 ./nfslxPegn
test completed successfully.

check for lost reply on non-idempotent requests
100 tries
```

```
test exclusive create.

test negative seek, you should get: read: Invalid argument
or lseek: Invalid argument

test rename

test truncate
truncate succeeded

test holey file support
Holey file test ok

second check for lost reply on non-idempotent requests
testing 50 idempotencies in directory "testdir"

test rewind support

test telldir cookies (slow)

write/read 30 MB file


test freesp and file size
fcntl(...F_FREESP...) not available on this platform.

Special tests complete

Starting LOCKING tests: test directory /mnt/nfstest/osmium.test (arg: -t)

Testing native pre-LFS locking

Creating parent/child synchronization pipes.

Test #1 - Test regions of an unlocked file.
Parent: 1.1  - F_TEST  [       0,       1] PASSED.
Parent: 1.2  - F_TEST  [       0,  ENDING] PASSED.
Parent: 1.3  - F_TEST  [       0,7fffffff] PASSED.
Parent: 1.4  - F_TEST  [       1,       1] PASSED.
Parent: 1.5  - F_TEST  [       1,  ENDING] PASSED.
Parent: 1.6  - F_TEST  [       1,7fffffff] PASSED.
Parent: 1.7  - F_TEST  [7fffffff,       1] PASSED.
Parent: 1.8  - F_TEST  [7fffffff,  ENDING] PASSED.
Parent: 1.9  - F_TEST  [7fffffff,7fffffff] PASSED.
```

```
Test #2 - Try to lock the whole file.
Parent: 2.0  - F_TLOCK [       0,  ENDING] PASSED.
Child:  2.1  - F_TEST  [       0,       1] PASSED.
Child:  2.2  - F_TEST  [       0,  ENDING] PASSED.
Child:  2.3  - F_TEST  [       0,7fffffff] PASSED.
Child:  2.4  - F_TEST  [       1,       1] PASSED.
Child:  2.5  - F_TEST  [       1,  ENDING] PASSED.
Child:  2.6  - F_TEST  [       1,7fffffff] PASSED.
Child:  2.7  - F_TEST  [7fffffff,       1] PASSED.
Child:  2.8  - F_TEST  [7fffffff,  ENDING] PASSED.
Child:  2.9  - F_TEST  [7fffffff,7fffffff] PASSED.
Parent: 2.10 - F_ULOCK [       0,  ENDING] PASSED.

Test #3 - Try to lock just the 1st byte.
Parent: 3.0  - F_TLOCK [       0,       1] PASSED.
Child:  3.1  - F_TEST  [       0,       1] PASSED.
Child:  3.2  - F_TEST  [       0,  ENDING] PASSED.
Child:  3.3  - F_TEST  [       1,       1] PASSED.
Child:  3.4  - F_TEST  [       1,  ENDING] PASSED.
Parent: 3.5  - F_ULOCK [       0,       1] PASSED.

Test #4 - Try to lock the 2nd byte, test around it.
Parent: 4.0  - F_TLOCK [       1,       1] PASSED.
Child:  4.1  - F_TEST  [       0,       1] PASSED.
Child:  4.2  - F_TEST  [       0,       2] PASSED.
Child:  4.3  - F_TEST  [       0,  ENDING] PASSED.
Child:  4.4  - F_TEST  [       1,       1] PASSED.
Child:  4.5  - F_TEST  [       1,       2] PASSED.
Child:  4.6  - F_TEST  [       1,  ENDING] PASSED.
Child:  4.7  - F_TEST  [       2,       1] PASSED.
Child:  4.8  - F_TEST  [       2,       2] PASSED.
Child:  4.9  - F_TEST  [       2,  ENDING] PASSED.
Parent: 4.10 - F_ULOCK [       1,       1] PASSED.

Test #5 - Try to lock 1st and 2nd bytes, test around them.
Parent: 5.0  - F_TLOCK [       0,       1] PASSED.
Parent: 5.1  - F_TLOCK [       2,       1] PASSED.
Child:  5.2  - F_TEST  [       0,       1] PASSED.
Child:  5.3  - F_TEST  [       0,       2] PASSED.
Child:  5.4  - F_TEST  [       0,  ENDING] PASSED.
Child:  5.5  - F_TEST  [       1,       1] PASSED.
Child:  5.6  - F_TEST  [       1,       2] PASSED.
Child:  5.7  - F_TEST  [       1,  ENDING] PASSED.
Child:  5.8  - F_TEST  [       2,       1] PASSED.
Child:  5.9  - F_TEST  [       2,       2] PASSED.
```

```
Child:  5.10 - F_TEST  [       2,  ENDING] PASSED.
Child:  5.11 - F_TEST  [       3,       1] PASSED.
Child:  5.12 - F_TEST  [       3,       2] PASSED.
Child:  5.13 - F_TEST  [       3,  ENDING] PASSED.
Parent: 5.14 - F_ULOCK [       0,       1] PASSED.
Parent: 5.15 - F_ULOCK [       2,       1] PASSED.

Test #6 - Try to lock the MAXEOF byte.
Parent: 6.0  - F_TLOCK [7fffffff,       1] PASSED.
Child:  6.1  - F_TEST  [7ffffffe,       1] PASSED.
Child:  6.2  - F_TEST  [7ffffffe,       2] PASSED.
Child:  6.3  - F_TEST  [7ffffffe,  ENDING] PASSED.
Child:  6.4  - F_TEST  [7fffffff,       1] PASSED.
Child:  6.5  - F_TEST  [7fffffff,       2] PASSED.
Child:  6.6  - F_TEST  [7fffffff,  ENDING] PASSED.
Child:  6.7  - F_TEST  [80000000,  ENDING] PASSED.
Child:  6.8  - F_TEST  [80000000,       1] PASSED.
Child:  6.9  - F_TEST  [80000000,7fffffff] PASSED.
Child:  6.10 - F_TEST  [80000000,80000000] PASSED.
Parent: 6.11 - F_ULOCK [7fffffff,       1] PASSED.

Test #7 - Test parent/child mutual exclusion.
Parent: 7.0  - F_TLOCK [       0,       4] PASSED.
Parent: Wrote ’aaaa’ to testfile [ 0, 4 ].
Parent: Now free child to run, should block on lock.
Parent: Check data in file to insure child blocked.
Parent: Read ’aaaa’ from testfile [ 0, 4 ].
Parent: 7.1  - COMPARE [       0,       4] PASSED.
Parent: Now unlock region so child will unblock.
Parent: 7.2  - F_ULOCK [       0,       4] PASSED.
Child:  7.3  - F_LOCK  [       0,       4] PASSED.
Child:  Write child’s version of the data and release lock.
Parent: Now try to regain lock, parent should block.
Child:  Wrote ’bbbb’ to testfile [ 0, 4 ].
Child:  7.4  - F_ULOCK [       0,       4] PASSED.
Parent: 7.5  - F_LOCK  [       0,       4] PASSED.
Parent: Check data in file to insure child unblocked.
Parent: Read ’bbbb’ from testfile [ 0, 4 ].
Parent: 7.6  - COMPARE [       0,       4] PASSED.
Parent: 7.7  - F_ULOCK [       0,       4] PASSED.

Test #8 - Rate test performing lock/unlock cycles.
Parent: Performed 1000 lock/unlock cycles in 710 msecs. [169014 lpm].

Test #10 - Make sure a locked region is split properly.
```

```
Parent: 10.0  - F_TLOCK [       0,       3] PASSED.
Parent: 10.1  - F_ULOCK [       1,       1] PASSED.
Child:  10.2  - F_TEST  [       0,       1] PASSED.
Child:  10.3  - F_TEST  [       2,       1] PASSED.
Child:  10.4  - F_TEST  [       3,  ENDING] PASSED.
Child:  10.5  - F_TEST  [       1,       1] PASSED.
Parent: 10.6  - F_ULOCK [       0,       1] PASSED.
Parent: 10.7  - F_ULOCK [       2,       1] PASSED.
Child:  10.8  - F_TEST  [       0,       3] PASSED.
Parent: 10.9  - F_ULOCK [       0,       1] PASSED.
Parent: 10.10 - F_TLOCK [       1,       3] PASSED.
Parent: 10.11 - F_ULOCK [       2,       1] PASSED.
Child:  10.12 - F_TEST  [       1,       1] PASSED.
Child:  10.13 - F_TEST  [       3,       1] PASSED.
Child:  10.14 - F_TEST  [       4,  ENDING] PASSED.
Child:  10.15 - F_TEST  [       2,       1] PASSED.
Child:  10.16 - F_TEST  [       0,       1] PASSED.

Test #11 - Make sure close() releases the process's locks.
Parent: 11.0  - F_TLOCK [       0,  ENDING] PASSED.
Parent: Closed testfile.
Child:  11.1  - F_TLOCK [       0,  ENDING] PASSED.
Child:  11.2  - F_ULOCK [       0,  ENDING] PASSED.
Parent: 11.3  - F_TLOCK [      1d,     5b7] PASSED.
Parent: 11.4  - F_TLOCK [    2000,      57] PASSED.
Parent: Closed testfile.
Child:  11.5  - F_TLOCK [       0,  ENDING] PASSED.
Child:  11.6  - F_ULOCK [       0,  ENDING] PASSED.
Parent: Wrote '123456789abcdef' to testfile [ 0, 16 ].
Parent: 11.7  - F_TLOCK [       0,  ENDING] PASSED.
Parent: Wrote '123456789abcdef' to testfile [ 13, 16 ].
Parent: Closed testfile.
Child:  11.8  - F_TLOCK [       0,  ENDING] PASSED.
Child:  11.9  - F_ULOCK [       0,  ENDING] PASSED.
Parent: Wrote '123456789abcdef' to testfile [ 0, 16 ].
Parent: 11.10 - F_TLOCK [       0,  ENDING] PASSED.
Parent: Truncated testfile.
Parent: Closed testfile.
Child:  11.11 - F_TLOCK [       0,  ENDING] PASSED.
Child:  11.12 - F_ULOCK [       0,  ENDING] PASSED.

Test #12 - Signalled process should release locks.
Child:  12.0  - F_TLOCK [       0,  ENDING] PASSED.
Parent: Killed child process.
Parent: 12.1  - F_TLOCK [       0,  ENDING] PASSED.
```

```
Test #13 - Check locking and mmap semantics.
Parent: 13.0  - F_TLOCK [     ffe,  ENDING] PASSED.
Parent: 13.1  - mmap [       0,    1000] WARNING!
Parent: **** Expected EAGAIN, returned success...
Parent: 13.2  - F_ULOCK [       0,  ENDING] PASSED.
Parent: unmap testfile.
Parent: 13.3  - mmap [       0,    1000] PASSED.
Parent: 13.4  - F_TLOCK [     ffe,  ENDING] PASSED.

** PARENT pass 1 results: 44/44 pass, 1/1 warn, 0/0 fail (pass/total).

**  CHILD pass 1 results: 64/64 pass, 0/0 warn, 0/0 fail (pass/total).
Congratulations, you passed the locking tests!

All tests completed
```