Introduction to the Connectathon NFS Testsuite

The Connectathon testsuite is available for downloading. The suite is available in two formats:

- nfstests.tar.gz for Unix clients (90 KB).
- nfstests.zip for PC clients (210 KB).

Last update: Wednesday, December 31st, 2003.

The test directories contain programs that can be used to test an implementation of the NFS Protocol. The tests run on a UNIX client and test server and client functions. They are divided into four groups:

- **basic** - basic file system operations tests
- **general** - general file system tests
- **special** - tests that poke certain common problem areas
- **lock** - tests that exercise network locking

This document is divided into five sections. The first section is the introduction, which you are reading now. That is followed by a description of what you have to do before you run the testsuites on your machine. Then comes a description of how the testsuites are run in general followed by a description of how they are used at Connectathon. The last section describes what each test does in detail.

This testsuite should run on both BSD and System V based systems. The System V Release 3 port of the Connectathon Testsuite is provided courtesy of the Lachman Technology, Incorporated, 1901 N. Naper Blvd., Naperville, IL. 60563.

Preparing to run the Testsuites

To prepare to run the testsuites on your machine, change directories to the highest level testsuite directory (it should be the same one that contains this README file), and type "make" to compile the test programs. If you are not sure you are in the correct directory, type "ls -CF" and you should see the following files and directories:

```
Makefile        basic/          lock/           tests.h
README          domount.c       runtests        tests.init
READWIN.txt     general/        server          tools/
Testitems       getopt.c        special/        unixdos.h
```

The "server" script uses "getopt". A source file of a public-domain version of "getopt" is included in the directory. The Makefile will compile it for you.

The tests are configured according to parameters found in the script, tests.init. It contains various definitions for commands and parameters used by the various Makefiles and shell scripts. This file should be checked and then perhaps modified to correctly match your system. In particular, the values of "MOUNTCMD", "UMOUNTCMD", "PATH", "CFLAGS", and "LIBS" should be checked and set correctly. There are several sets of suggested values which may be used as possible starting places.

Two special targets are included in the Makefiles: copy and dist. The command

make copy DESTDIR=*path*

where *path* is the absolute name of a directory, will cause the compiled tests to be copied to *path*. The command

make dist DESTDIR=*path*

where *path* is the absolute name of a directory, will copy the test sources to *path*. DESTDIR must be specified on the make command line when making either of these targets.

Modifications may be required so the programs compile on your machine. If that is so, we would like to know what they are so that we can incorporate them into our distribution.

When defaults are used, the test programs expect the directory, /server, to exist on the server. The test driver will use the directory /mnt/'server_name' on the client, creating it first if necessary (where 'server_name' is the name of the server you are testing against). These defaults can be overridden at run time. Directions for doing this are contained in the next section.

How to run the Testsuites

There are two ways to run the tests: use the server shell script or mount, run the tests yourself, and unmount. We recommend you use the server script to run the tests.

The server script:

The server script executes the basic and general tests. (It runs the special tests only if use the -s option.) It is set up to mount, run tests using the runtests program, and unmount. It will attempt to unmount anything mounted on the mount point before attempting to mount the server file system. If a test fails, the run is aborted and the file system is left mounted to assist in troubleshooting the failure.

The server script uses the domount program to mount and unmount the test file systems. Since mount can only be executed by root, domount must have root permission. The Makefile will attempt to setuid the domount program to root. The server script can be run as a nonprivileged user. Alternately, you may login as root before you run server.

```
server [-a|-b|-g|-s|-l] [-f|-t] [-n] [-o mnt_options] [-p server_path] [-m
mntpoint] [-N numpasses] server_name

-a|-b|-g|-s|-l - will be passed on to the runtests scripts.  This argument
              is optional.  The default is read from the initialization
              file, tests.init.  The variable, TEST, contains this
              argument.
              This argument selects which tests to run:
                -a      run basic, general, special, and lock tests
                -b      run basic tests only
```

```
                -g      run general tests only
                -s      run special tests only
                -l      run lock tests only
-f|-t       - will be passed on to the runtests scripts.  This argument
              is optional.  The default is read from the initialization
              file, tests.init.  The variable, TESTARG, contains this
              argument.
              This argument selects how the basic tests are to be run:
                -f      a quick functionality test
                -t      extended test mode with timings
-n          - Don't perform the mkdir and rmdir operations to create
              and destroy the test directory.
-o mnt_options - will be passed on to the mount command.  This argument is
              optional.  The default is read from the initialization
              file, tests.init.  The variable, MNTOPTIONS, contains this
              argument.
-p server_path - specifies a directory on the server to mount.  This
              argument is optional.  The default is read from the
              initialization file, tests.init.  The variable, SERVPATH,
              contains this argument.
-m mntpoint    - specifies a mount point on your client. This argument is
              optional.  The default is read from the initialization
              file, tests.init.  The variable, MNTPOINT, contains this
              argument.
-N numpasses - will be passed to the runtests script.  This argument
              is optional.  It specifies the number of times to run
              through the tests.
server_name - the server you want to exercise.  This is the only
              required argument.
```

The test programs create a sub-directory in the mntpoint directory with the name, 'hostname'.test, (where 'hostname' is the name of the machine on which you run the tests). This name can not be overridden if you use the server script although it can be if you use runtests directly.

Example: (the client machine is eddie)

```
eddie% server -o hard,intr,rw slartibartfarst
Start tests on path /mnt/slartibartfast/eddie.test [y/n]? y
<output from tests>
         :
         :
All tests completed
eddie%
```

See the script for more details.

Run tests yourself:

There is a runtest script in the highest level directory (the master runtests) which uses tests.init to set up the test environment and then executes the runtest scripts in the basic, general, and/or special sub-directories.

```
runtests [-a|-b|-g|-s|-l] [-f|-n|-t] [-N numpasses] [test-directory]

-a             - Run the basic, general, special, and lock tests.  This
                 is the default.
-b             - Run the basic tests.
-g             - Run the general tests.
-s             - Run the special tests.
-l             - Run the lock tests.
-f             - Set parameters for a quick functional test.  It
                 applies only to basic tests.
-n             - Suppress directory operations (mkdir and rmdir) on the
                 test-directory.  See descriptions of basic tests for
                 more details.
-t             - Run full-length test with running time statistics. It
                 only applies to basic tests.  This is the default mode
                 for the basic tests.
-N numpasses   - Run the tests "numpasses" times.
test-directory - The name of test directory that the test programs
                 create on the client.  runtests executes the basic
                 tests in place and they work on the test directory.
                 The general tests are copied over to the test
                 directory and executed there.  When the -n flag is
                 used, the test directory is assumed to already exist.

                 The default test-directory is
                 /mnt/'servername'/'hostname'.test (where 'servername'
                 is the name of the server being tested, and
                 'hostname' is the name of the machine on which you
                 are running the tests).  There are three ways to
                 override the default test directory name.  One it to
                 put the test_directory on the command line.  Another
                 way is to set the environment variable, NFSTESTDIR,
                 equal to the directory name.  The command line method
                 overrides setting the environment variable.  The
                 third way can only be used for the tests in the basic
                 sub-directory.  There you can set the TESTDIR
                 variable in tests.h.  The command line and
                 environment variable both override this method.
```

Running the tests without mounting your NFS server on /mnt will run the tests locally (if /mnt is local disk). We recommend that you do this once to make sure the testsuites run properly before you use them to test NFS.

The runtests in the sub-directories, basic, general, and special, may be invoked with the same arguments as the master runtests if you wish to run each suite separately.

How to run the Testsuites at Connectathon

The tests should be run in the following order: basic, general, and special. The basic tests should be passed completely before other tests are attempted.

The NFS Test Suite should be run in three phases:

Phase 1 - Run test programs locally.

Phase 2 - Run the tests against a Sun.

Run them on your machine using the Sun as the server and then run them on the Sun using your machine as the server.

Phase 3 - NxN Testing.

Run the tests on your machine using every other machine as a server, one at a time. After the tests are successfully completed using a particular server, log that with the electronic board software provided. Check the electronic board to make sure that the tests run successfully on every other machine that uses your machine as a server.

Basic Test Descriptions

System and library calls that are used by the testsuites are included in parentheses. Look at the source if you are interested in how time statistics are recorded since that is not included in this description.

Many of the programs listed below have optional calling parameters that can be used to override existing parameters. These are not used at this time so they are not described.

Test1: File and Directory Creation Test

This program creates the test directory (mkdir) on the client and changes directories (chdir) to it, unless the -n flag is used in which case it simply changes directories to the test directory. Then it builds a directory tree N levels deep, where each directory (including the test directory) has M files and P directories (creat, close, chdir, and mkdir). For the -f option, N = 2, M = 2, and P = 2 so a total of six files and six directories are created. For other options, N = 5, M = 5, and P = 2. The files that are created are given names that begin with "file." and directories with names that begin with "dir.".

Test2: File and directory removal test

This program changes directory to the test directory (chdir and/or mkdir) and removes the directory tree (unlink, chdir, and rmdir) that was just created by test1. The number of levels, files, and directories, and the name prefixes, are the same as in test1.

This routine will not remove a file or directory that was not created by test1 and will fail if it finds one. It determines this by looking at the prefix on the name of the object it's trying to remove.

Test3: Lookups across mount point

This program changes directory to the test directory (chdir and/or mkdir) and gets the file status of the working directory (getwd or getcwd and stat). For the -f option, the getwd or getcwd is done once. For other options, 250 getcwds or getcwds are done.

Test4: setattr, getattr, and lookup

This program changes directory to the test directory (chdir and/or mkdir) and creates ten files (creat). Then the permissions are changed (chmod) and the file status is retrieved (stat) for each file. For the -f option, one chmod and stat on each file is done. For other options, 50 getcwds or getcwds and stats on each file are done.

Test4a: getattr, and lookup

This test exists but is not called as part of the testsuite. You can edit runtests in the basic directory so this test is called.

This program changes directory to the test directory (chdir and/or mkdir) and creates ten files (creat). Then the file status is retrieved (stat) for each file. For the -f option, the stat is done once per file. For other options, 50 stats are done per file.

Test5: read and write

This program changes directory to the test directory (chdir and/or mkdir) and then:

1. Creates a file (creat)
2. Gets status of file (fstat)
3. Checks size of file
4. Writes 1048576 bytes into the file (write) in 8192 byte buffers.

5. Closes file (close)
6. Gets status of file (stat)
7. Checks the size of the file

For the -f option, the file is created and written once. For other options, file is created and written 10 times.

Then the file is opened (open) and read (read) in 8192 byte buffers. It's contents are compared with what was written. The file is then closed (close).

Then the file is then re-opened (open) and re-read (read) before it is removed (unlink). For the -f option, this sequence is done once. For other options, this sequence is done 10 times.

Test5a: write

This test exists but is not called as part of the testsuite. You can edit runtests in the basic directory so this test is called.

This program changes directory to the test directory (chdir and/or mkdir) and then:

1. Creates a file (creat)
2. Gets status of file (fstat)
3. Checks size of file
4. Writes 1048576 bytes into the file (write) in 8192 byte buffers.
5. Closes file (close)
6. Gets status of file (stat) <LI
7. > Checks the size of the file

For the -f option, the file is created and written once. For other options, file is created and written 10 times.

Test5b: read

This test exists but is not called as part of the testsuite. You can edit runtests in the basic directory so this test is called.

The file created in test5a is opened (open) and read (read) in 8192 byte buffers. It's contents are compared with what was written. The file is then closed (close) and removed (unlink).

For the -f option, the file is opened and read once. For other options, file is created and written 10 times.

Test6: readdir

This program changes directory to the test directory (chdir and/or mkdir) and creates 200 files (creat). The current directory is opened (opendir), the beginning is found (rewinddir), and the directory is read (readdir) in a loop until the end is found. Errors flagged are:

1. No entry for "."
2. No entry for ".."
3. Duplicate entry
4. Filename that doesn't begin with "file."
5. The suffix of the filename is out of range
6. An entry is returned for an unlinked file. (This error can only be found when the test is run with an option other than -f. For other options the rewinddir/readdir loop is done 200 times and a file is unlinked each time).

The directory is then closed (closedir) and the files that were created are removed (unlink).

Test7: link and rename

This program changes directory to the test directory (chdir and/or mkdir) and creates ten files. For each of these files, the file is renamed (rename) and file statistics are retrieved (stat) for both the new and old names. Errors that are flagged are:

1. Old file still exists
2. New file doesn't exist (can't stat)
3. The new file's number of links doesn't equal one

Then an attempt is made to link the new file to it's old name (link) and file stats are again retrieved (stat). An error is flagged if:

1. Can't link
2. Stats on new file can't be retrieved after link
3. The new file's number of links doesn't equal two
4. Stats on old file can't be retrieved after link
5. The old file's number of links doesn't equal two

Then the new file is removed (unlink) and file stats are retrieved for the old file (stat). An error is flagged if:

1. Stats on old file can't be retrieved after unlink
2. The old file's number of links doesn't equal one

For the -f option, the rename/link/unlink loop is done once for each file. For other options, the rename/link/unlink loop is done 10 times for each file.

Any files that remain at the end of the test are removed (unlink).

Test7a: rename

This test exists but is not called as part of the testsuite. You can edit runtests in the basic directory so this test is called.

This program changes directory to the test directory (chdir and/or mkdir) and creates ten files. For each of these files, the file is renamed (rename) and file statistics are retrieved (stat) for both the new and old names. Errors that are flagged are:

1. Old file still exists
2. New file doesn't exist (can't stat)
3. The new file's number of links doesn't equal one

The file is then renamed back to its original name and the same tests are applied.

For the -f option, the rename/rename loop is done once for each file. For other options, the rename/rename loop is done 10 times for each file.

Any files that remain at the end of the test are removed (unlink).

Test7b: link

This test exists but is not called as part of the testsuite. You can edit runtests in the basic directory so this test is called.

This program changes directory to the test directory (chdir and/or mkdir) and creates ten files. A link (link) is done for each of these files and file stats are retrieved for the old and new files (stat). An error is flagged if:

1. Can't link
2. Stats on either file can't be retrieved after link
3. The either file's number of links doesn't equal two

This is followed by an unlink (unlink) of the new file. An error is flagged if:

1. Stats on the old file can't be retrieved after unlink
2. The old file's number of links doesn't equal one

For the -f option, the link/unlink loop is done once for each file. For other options, the link/unlink loop is done 10 times for each file.

Any files that remain at the end of the test are removed (unlink).

Test8: symlink and readlink

NOTE: Not all operating systems support symlink and readlink. If the errno, EOPNOTSUPP, is returned during test8, the test will be counted as passing. For clients not supporting S_IFLNK, the test will not be attempted.

This program changes directory to the test directory (chdir and/or mkdir) and makes 10 symlinks (symlink). It reads (readlink), and gets statistics for (lstat) each, and then removes them (unlink). Errors flagged are:

1. Unsupported function
2. Can't get statistics (lstat failed)
3. The mode in the stats is not symlink
4. The value of the symlink is incorrect (returned from readlink)
5. The linkname is wrong
6. The unlink failed

For the -f option, the symlink/readlink/unlink loop is done for each symlink. For other options, the symlink/readlink/unlink loop is done 20 times for each symlink.

Test9: statfs

This program changes directory to the test directory (chdir and/or mkdir) and gets the file system status on the current directory (statfs). For the -f option, the statfs is done once. For other options, the statfs is done 1500 times.

Other Tests

GENERAL
: General tests to look at server loading. Runs a small compile, tbl, nroff, a large compile, four simultaneous large compiles, and make.

SPECIAL
: Information specific to the special tests.

The special directory is set up to test special problems that have come up in the past. These tests are meant to be advisory, things to watch out for. It is not required that you "pass" these tests but we strongly suggest that you do.

The tests try to:

- Check for proper open/unlink operation
- Check for proper open/rename operation
- Check for proper open/chmod 0 operation
- Check for lost reply on non-idempotent requests
- Test exclusive create
- Test negative seek
- Test rename

LOCK

The lock directory contains a test program which can be used to test the kernel file and record locking facilities. This is done to test the network lock manager.

The test program contains 13 sets of locking tests. They test basic locking functionality.

By default, mandatory locking is not tested. Mandatory locking is generally *not* supported on NFS files.

Miscellaneous

'Testitems' is a list of NFS functionality that can be used for reference.

Programs in 'tools' are provided for your use as you see fit. Please feel free to add to this (or any other) directory! If you do, please make sure that `siddheshwar.mahesh@sun.com` gets a copy so we can add it to the master test distribution.

Other Information

Changes for 2004 include the following:

1. Fix lock/tlock.c to be consistent about when to use stdarg and when to use varargs; reported by Samuel Sha.
2. Change "make all" so that the various "runtests" scripts have the execute bit set; reported by Erik Deumens.
3. Removed some lint; from James Peach.
4. Irix 6.5.19 support from James Peach.
5. The "server" script now exports `MNTOPTIONS`, so that options that are added to "server" can be detected by the rest of the suite. From Chuck Lever.
6. The tests now correctly check for errors returns from mmap(). From David Robinson.
7. MacOS X support from Mike Mackovitch.
8. tests.init now includes a `CC=` line for Linux, in case your distribution doesn't include "cc". Reported by Rodney Brown.

9. Changes for AIX, from Erik Deumens.
10. Changes for the latest Tru64 Unix, from Eric Werme.
11. The general tests should be more robust in the face of errors from make(1). Based on comments from Chuck Lever and a patch from Mike Mackovitch.
12. The "make lint" target for the basic tests now includes subr.c.
13. Improvements to special/bigfile2:
    - error messages now print the complete low-order word (from Mike Mackovitch.
    - the test file is opened with `O_SYNC`, so that problems are detected right away.
14. Fix to special/op_chmod so that it uses `CHMOD_NONE` instead of 0. From Pascal Schmidt.