About:

This document is aimed to provide detail and structured information about NFSv4, ACLs and contains description of the test cases (implemented in Python) and my work about discovery NFSv4 and ACLs.

Creator:

A.V.Nesterovich@gmil.com

References:

Title	Reference
RFC7530 Network File System (NFS) Version 4 Protocol	https://www.rfc-editor.org/rfc/rfc7530.txt
RFC7531 Network File System (NFS) Version 4 External Data Representation Standard (XDR) Description	https://www.rfc-editor.org/rfc/rfc7531.txt
Mapping Between NFSv4 and Posix Draft ACLs	http://www.citi.umich.edu/projects/nfsv4/rfc/draft-ietf-nfsv4-acl-mapping-05.txt
acl - Access Control Lists (Linux man page) *POSIX	http://linux.die.net/man/5/acl
exports - NFS server export table (Linux man page)	http://linux.die.net/man/5/exports
exportfs - maintain table of exported NFS file systems(Linux man page)	http://linux.die.net/man/8/exportfs
rpc.mountd - NFS mount daemon (Linux man page)	http://linux.die.net/man/8/mountd
getfacl - get file access control lists (Linux man page)	http://linux.die.net/man/1/getfacl
setfacl - set file access control lists (Linux man page)	http://linux.die.net/man/1/setfacl

NFS4

Info

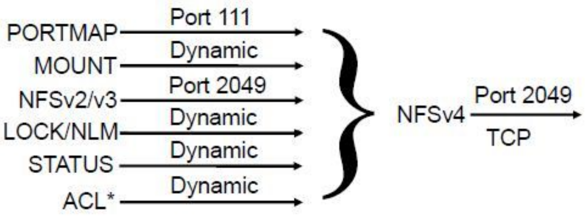
NFS is a UNIX protocol for large scale client/server file sharing. It is analogous to the server Message Block (SMB) and Common Internet File System (CIFS) protocols on Microsoft Windows. The Network File System Version 4 is a distributed filesystem protocol which owes heritage to NFSv2 and NFSv3. Unlike previous versions of NFS the present version(NFSv4) supports traditional file access while integrating support for file locking and mount protocol.

NFSv4 is the successor of NFSv3. It has been designed to work on a LAN or over the Internet.

NFS v4 provides the following benefits over NFSv3 or earlier NFS versions:

- **advanced security management** (mandates security and ACLs); Kerberos; SPKM; LIPKEY;

- **firewall friendly** (NFS v4 work by default works over TCP): the use ofportmapper dishing out arbitrary ports made it difficult for the firewall. NFSv4 changed that by consolidating most of the TCP/IP services into well-known ports which the security administrator can define in the firewall



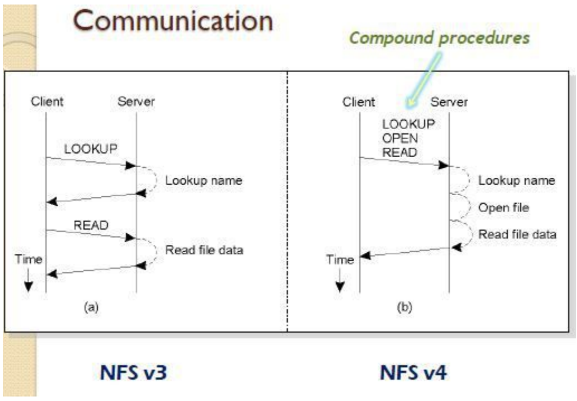
- **advanced and aggressive cache management;**

- **non Unix compatibility (Windows);**

- **easy to administer (Replication, migration);**

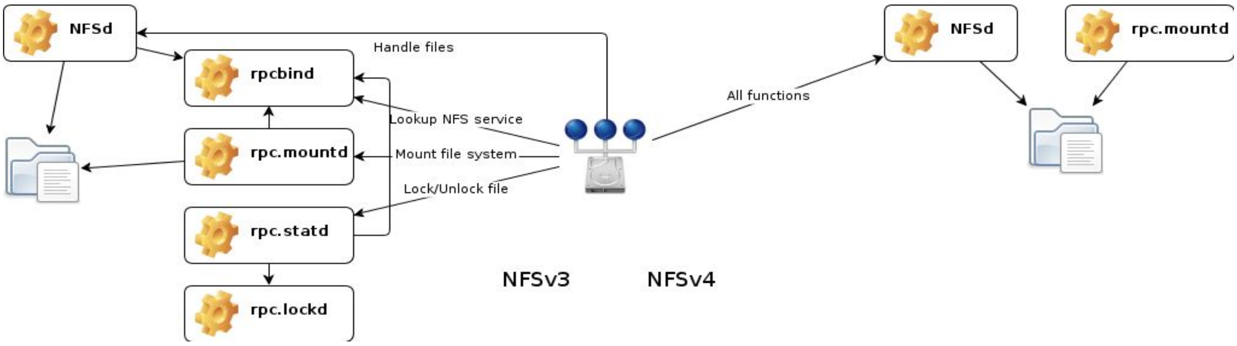
- **crash recovery (Client and server sides);**

- **performance improvements:** one key enhancement is the introduction of the COMPOUND RPC procedure which allows the NFS client to group together a bunch of file operations into a single request to the NFS server. This not only reduces the network round-trip latency, but also reduces the small little chatters of the smaller file operations.



- **stateful:** NFSv3 is stateless and it does not maintain the state of the NFS clients. NFSv4 is stateful and implements a mandatory locking and delegation mechanisms. Leases for locks from the servers to the clients was introduced. A lease is a time-bounded grant for the control of the state of a file and this is implemented through locks.

The NFSv3 and NFSv4 protocols are not compatible. A NFSv4 client cannot access a NFSv3 server, and vice versa. However, in order to simplify migrations from NFSv3 to NFSv4, both NFSv3 and NFSv4 services are launched by the command: **rpc.nfsd**. In the case of NFSv3 and NFSv4 clients simultaneously accessing the same server, one must be aware that two different file systems are used: there is no backward support to NFSv3 by the NFSv4 server. In order to ensure a better reliability over the Internet, NFSv4 only uses TCP. To help NFS setup for internet use, one unique network port is used on NFSv4. This predetermined port is fixed. The default is **port 2049**. With NFSv4, mount and locking services have been integrated in the NFS daemon itself.



Compare NFSv3 and NFSv4 implementations

The NFSv4 server and clinet work **without the portmap, rpc.lockd, rpc.statd** daemons. The **rpc.mountd daemon is still required** on the server.

Since NFSv4 no longer utilizes the **rpc.mountd** protocol as was used in NFSv2 and NFSv3, the mounting of file systems has changed. An NFSv4client now has the ability to see all of the exports served by the NFSv4server as a single file system, called the **NFSv4 pseudo-file system**. On Red Hat Enterprise Linux, the pseudo-file system is identified as a single, real file system, identified at export with the fsid=0 option.

A NFSv4 client communicates with corresponding NFSv4 Server via Remote Procedure Calls (RPS's). The client sends a request and gets a reply from the server. A NFSv4 server can only provide/export a single, hierarchical file system tree. If a server has to share more than one logical file system tree, the single trees are integrated in a new virtual root directory. This construction, called pseudo file system, is the one which is provided/exported to clients.

ACL

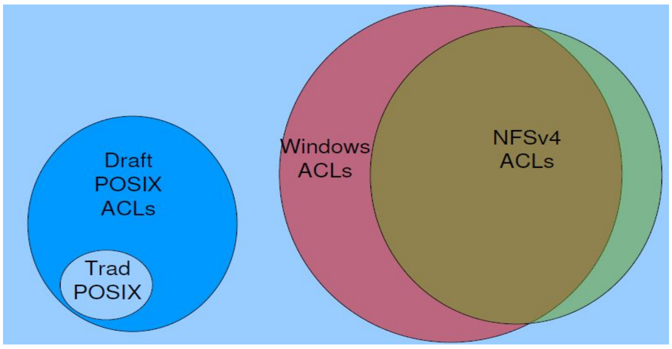
General Info

POSIX is a family of standards, specified by the IEEE, to clarify and make uniform the application programming interfaces (and ancillary issues, such as commandline shell utilities) provided by Unix-y operating systems.

ACL = Access Control List (ACLs allow a sysadmin to express nontrivial rules defining access control on objects (e.g. files or directories))

Option	Access Model In General	Access Models For Filesystems
Subject	entity which performs actions	process
Object	entity on which actions are performed	inode, i.e. file or directory or other object
Algorithm	**(subjectcredentials, objectpermissions, actionrequested)** 1) definition of format for subject credentials 2) definition of format for object permissions set of actions a subject can perform	**might use the permissions of the parent directory** 1) subject credentials = an owner, some groups 2) object permissions = an owner, an owning group,a POSIX mode and/or some kind of ACL actions e.g. Read, Write, Execute, Delete, Change Owner

Access Models For Filesystems:

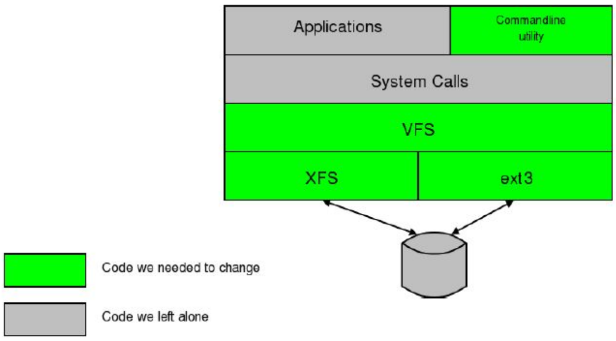


Model	Description
Traditional POSIX	Owners & groups identified by UIDs, GIDs / Locally mapped on each system (e.g. /etc/passwd, NIS (Network Information System, LDAP (Lightweight Directory Access Protocol)) / 3 bit access mask / Read (r), Write (w), Execute (x) / Most actions are mapped to one of these 3 / − exceptions, e.g. Delete => Write on the parent directory / Subjects classified into one of 3 classes / − Owner = the subject's owner matches the object's, or / − Group = one of the subject's groups matches the object's owning group, or / − Other = none of the above / 1 access mask per class = 9 bits / setuid, setgid, sticky bits = 12 bits mode (Total) / Classify subject by matching UIDs and GIDs / - Use class to choose one of the 3 access masks / - If desired bit is set access is allowed, else denied
Draft POSIX ACLs	Draft POSIX extensions 1003.1e and 1003.2c / − never ratified / − but implemented several times

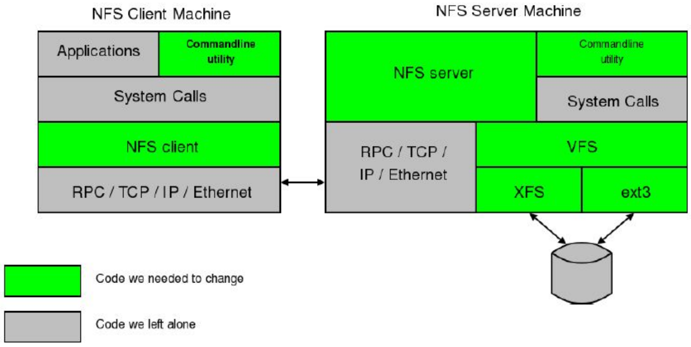
	− simply extends POSIX to allow more entries / Users & groups UIDs/GIDs (same) / 3 bit access mask (same) / 3 special classes (same) / − Owner (tag ACL_USER_OBJ), Group (ACL_GROUP_OBJ), Other (ACL_OTHER) / 1 entry per each special class / Any number of additional entries (ACEs) − General user (ACL_USER), general group (ACL_GROUP) / − Total between 3 .. _POSIX_ACL_ENTRIES_MAX (>= 16) entries / − Order not significant / Entries only allow access, never deny access / − Access not explicitly allowed is implicitly denied / If subject UID matches an ACL_USER or ACL_USER_OBJ entry, / use that / Else if subject GIDs match an ACL_GROUP_OBJ or ACL_GROUP entry, use that / Else use the ACL_OTHER entry
Windows NT ACLs	From Microsoft / − Implemented in Windows NT / − Minor changes in subsequent releases Users & groups identified by SIDs (Security Identifiers) / − Like a variable-length enormous binary UID but with global scope / − e.g. S152110043363481177238915682003330512 / 14 access mask bits / − ReadData/ListFolder, WriteData/CreateFile, AppendData/CreateFolder, /ReadExtendedAttributes, WriteExtendedAttributes, Execute/TraverseFolder, / DeleteChild, ReadAttributes, WriteAttributes, Delete, ReadPermissions, / WritePermissions, TakeOwnership, Synchronize / Variable number of ACEs / − Up to 64K size = ~1800 ACEs / − Entry has a SID, a type, some flags, and an access mask / − Order significant / 3 useful special classes (WellKnown SIDs) − Creator Owner, Creator Group, Everyone /Several less helpful WellKnown SIDs / − Interactive, Network, Dialup, Batch, Anonymous, Authenticated, Service etc / Any number of other SIDs − general user, general group / Entry type / − AccessAllowed = subject is allowed the access bits − AccessDenied = subject is denied the access bits / − SystemAudit = wacky audit stuff Entry flags / − INHERITED, INHERIT_ONLY, CONTAINER_INHERIT, OBJECT_INHERIT, / NO_PROPAGATE_INHERIT − supports inheritance / − SUCCESSFUL_ACCESS, FAILED_ACCESS − wacky audit stuff / ACL flags / − Actually in the containing Security Descriptor / − AUTO_INHERITED, DEFAULTED, PROTECTED / Loop through each entry / − stop if entry's SID matches any of subject's SIDs / If matching entry is Allow, access allowed / If matching entry is Deny, access denied / If no matching entry, access denied / TODO:reexpress
NFSv4 ACLs	Defined in NFSv4 standards / Tries to make the Windows access model usable over NFS / − without admitting it's Windows / − obviously nobody implemented it before writing the RFC / − architecture very similar to Windows, differs only in details / Users & groups identified by strings − “user@domain” or “group@domain” / − for transmission; systems expected to map these to some other local form/ like UIDs / 14 access mask bits / − Binary values identical to Windows − Names and semantics...similar...to Windows / − NFSv4.1 adds 2 more which have no equivalent in Windows / 3 useful special classes (whos) / − OWNER@, GROUP@, EVERYONE@ − Nearly identical to Windows / Several unhelpful special classes / − Blindly copied from Windows − But much less helpful in an NFS context / Variable number of other entries TODO:rephrases − General user, general group / Any number of entries / − No defined limit / Entry type − Blindly copied from Windows / − Including audit / Per-entry flags / − Blindly copied from Windows / − Added ACE4_IDENTIFIER_GROUP to tell apart user & group whos / Per-ACL flags − Blindly copied from Windows / − Not in NFSv4, added in 4.1

Implement NFSv4 ACLs in:

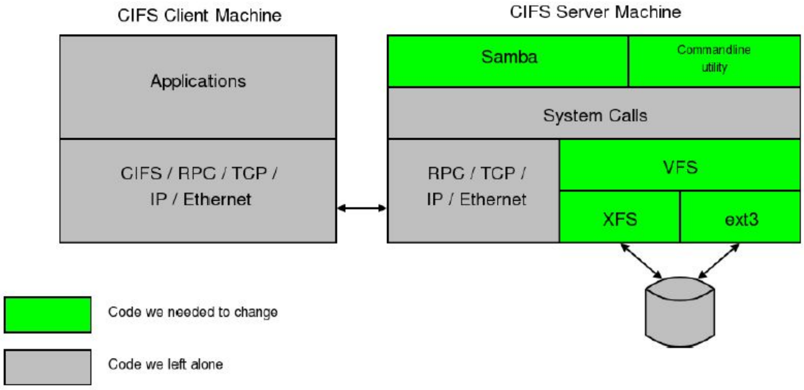
NFS4 ACL (Server)	NFS4 ACL (Client)
Server assumes underlying filesystem does POSIX ACLs − Converts to POSIX ACL when client sets an ACL − Converts from POSIX ACL when client gets an ACL − In general this conversion is lossy − Samba is doing (hopefully) the same conversion in userspace	Client presents ACLs in an unexpected manner − Nonstandard Extended Attribute, formatted as NFSv4 XDR − Need special utilities to set, print − These utilities are different to what's used on the server − Problems with cp, tar
NFSv4 ACLs are more expressive: − e.g. Deny ACEs, inheritance control / − you might learn to like that extra power / **Can a mixed Windows/Linux environment:** − with global access policies / − and files being shared between both clients / − over both NFS and CIFS protocols	



Architecture: Local Applications



Architecture: Using NFS



Architecture: Using CIFS

NFSv4 ACLs vs POSIX

ACE4_SYNCHRONIZE access bit

− obviously makes no sense on POSIX

ACE4_{READ,WRITE}_NAMED_ATTR access bits

− make no sense either, but rather less obviously!

Preserving more obscure corners of POSIX behaviour

– Sticky bit. CAP_FOWNER, CAP_CHOWN. Restricted_chown.

Doing chmod right: file_masks and the protocol

EVERYONE@ != POSIX Other class

– Far too easy to forget

Other whos (INTERACTIVE@ etc)

– Make no sense in NFS context; preserved but ignored

ACL Text Representation

List of ACEs, separated by whitespace

Each ACE is four fields separated by colons

who:accessmask:flags:type (e.g. accounts:rwax:g:allow)

1) who

2) access mask – 1char abbrevs, any order

r = read_data/list_directory

w = write_data/add_file

a = append_data/add_subdirectory

x = execute / traverse_directory,

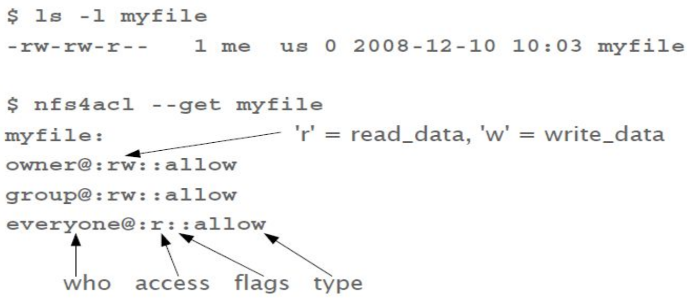
etc etc

3) Flags – 1char abbrevs, any order

g = who field names a group

4) Type

allow, deny



```
$ ls -l myfile
-rw-rw-r--   1 me   us 0 2008-12-10 10:03 myfile

$ nfs4acl --get myfile
myfile:
owner@:rw::allow
group@:rw::allow
everyone@:r::allow
```

Test cases (task)

Intro

Task verifies several candidate abilities including:

- ability to study independently by learning POSIX file systems standard

- perform data analysis by selecting important information

- design test cases and prepare documentation

- implement code based on design

Test Task

Design a set of test cases for owner / permission / content modification testing of NFSv4file system. Implement designed test cases as testing application (test suite). E.g, all tests are stored in “tests” folder and there is “main” file which run all tests and produces an output.

The results of the task are:

1. Test documentation. Use the following format:

- Name of test case

- Description

- Steps

- Expected result of each step

2. Source code of test suite

3. Logs of the latest successful tests execution

Test case example:

- Test name: Change file attributes to disable run, enable it, disable again.

- Description: test verifies that after several disable, enable actions permissions set to last value.

Acceptance criteria:

1. At least 6 test cases have to be created

2. At least 2 test cases for ACL management verification (optional)

3. Test documentation

4. Use one of the following scripting language:

a. Python (preferable) b. Ruby c. Perl (in OOP style)

5. Test Suite has to prepare and clean environment

6. Keep logs in log file, Short summary should be printed at the end of testing. E.g.:

TC001: Passed TC002: Failed TC003: Passed

7. Should be executable at any Linux-Like system

8. Please use comments in the code

Tests cases (Implementation)

Goal:

To develop the test suite and create documentation (subject NFSv4 with ACL support test automatization for Linux-like systems [server-client sides])

Test environment

Hostname	IP	Software	Description
fedora	192.168.100.182 (LAN)	Fedora 23 Workstation x86-x64 (ext4) **rsync**	Python 2.7, IDE Pycharm (create tests) Client NFSv4 (run tests)
nestu	Dynamic IP (WAN)	Fedora 23 Workstation x86-x64 (ext4) **rsync**	Python 2.7, IDE Pycharm (create tests) Client NFSv4 (run tests)
rhel	192.168.100.176 (LAN)	Red Hat Enterprise Linux Server release 7.2 (Maipo) (RHEL) x86-x64 (LVM, XFS)	Server NFSv4 (run tests)

Daemons NFS

	both sides	server side
user daemons	**rpc.idmapd** - This process provides NFSv4 client and server upcalls which map between on-the-wire NFSv4 names (which are strings in the form of user@domain) and local UIDs and GIDs. For idmapd to function with NFSv4, the /etc/idmapd.conf must be configured. This service is required for use with NFSv4.	**rpc.nfsd** - Allows explicit NFS versions and protocols the server advertises to be defined. It works with the Linux kernel to meet the dynamic demands of NFS clients, such as providing server threads each time an NFS client connects. This process corresponds to the nfs service. **rpc.mountd** - daemon is still needed to handle the exports, but is not involved with network communication anymore (in other words, the client connects directly with the NFS daemon).
kernel parts	NFSv4, RPC, XDR, TCP, IPv4	

Packets and tools NFS

Name	Description	Addition
nfs-utils	The nfs-utils package provides a daemon for the kernel NFS server and related tools, which provides a much higher level of performance than the traditional Linux NFS server used by most users.	-
nfs4-acl-tools	The nfs4-acl-tools packages provide utilities for managing NFSv4 Access Control Lists (ACLs) on files and directories mounted on ACL-enabled NFSv4 file systems. These updated packages fix the following bug.	-
libnfsidmap	Is a library holding mulitiple methods of mapping names to id's and visa versa, mainly for NFSv4.	-
showmount	show mount information for an NFS server	-

Configs NFS

File	Description	Comment
Server side		
/etc/exports	It is a main configuration file, controls which file systems are exported to remote hosts and specifies options. This file contains a list of entries; each entry indicates a volume that is shared and how it is shared.	There must be at least one entry with fsid=0. (this will be pseudo file system's /). **acl** option use.
/etc/sysconfig/nfs	This file is used to control which ports the required RPC services run on. Here the number of kernel threads, NFSv4 support and GSS security (kerberos) for NFS can be configured.	Not used on the project
/etc/idmapd.conf	It translates user and group ids into names, and to translate user and group names. Use to modify the default "Domain" to contain DNS domain name.	Not used on the project
Client side		
/etc/fstab	This file is used to control what file systems including NFS directories are mounted when the system boots.	**acl** option use
/etc/idmapd.conf	It translates user and group ids into names, and to translate user and group names. Use to modify the default "Domain" to contain DNS domain name.	Not used on the project

Useful commands NFS

Description (Action)	Client side (Comand)	Server side (Comand)
Umount all nfs mounts on client	umount -a -t nfs	-
Check all nfs mounts on client	mount | grep nfs	-
Reexport of shares files Display of shares files	-	exportfs -r exportfs -v
Check all registered RPC programs (nfs, portmapper, mountd, ...)	rpcinfo -p	rpcinfo -p
Check mount information on NFS server	showmount -e <server IP>	showmount -e <server IP>
Display statistics kept about NFS client and server activity	nfsstat	nfsstat
Get file access control lists. For each file, getfacl displays the file name, owner, the group, and the Access Control List (ACL). If a directory has a default ACL, getfacl also displays the default ACL. Non-directories cannot have default ACLs.	getfacl -R <dir>	getfacl -R <dir>
Set file access control lists	setfacl <options>	setfacl -R <options>

Install and settings of NFSv4 (*** without Kerberos)
Server-side (hostname: rhel)

1) Modify hosts file in order to resolve IP from hostname

```
vim /etc/hosts
192.168.100.176 rhel
192.168.100.182 fedora

ping fedora
PING fedora (192.168.100.182) 56(84) bytes of data.
64 bytes from fedora (192.168.100.182): icmp_seq=1 ttl=64 time=0.335 ms
```

2) In order to use ACLs enable mount option

```
vim /etc/fstab
dev/mapper/rhel_nfs-root /                  xfs    defaults,acl       0 0
mount -a
```

3) Check which loadable kernel modules NFS currently loaded

```
lsmod | grep nfs
nfs                 251815  0
fscache               64987  1 nfs
nfsd                302351  1
auth_rpcgss           59314  1 nfsd
nfs_acl              12837  1 nfsd
lockd               93572  2 nfs,nfsd
grace                13288  2 nfsd,lockd
sunrpc              300421  8 nfs,nfsd,auth_rpcgss,lockd,nfs_acl
```

4) Check and show information about NFS modules:

```
modinfo nfs
filename:      /lib/modules/3.10.0-327.13.1.el7.x86_64/kernel/fs/nfs/nfs.ko
modinfo nfsv3
filename:      /lib/modules/3.10.0-327.13.1.el7.x86_64/kernel/fs/nfs/nfsv3.ko
modinfo nfsv4
filename:      /lib/modules/3.10.0-327.13.1.el7.x86_64/kernel/fs/nfs/nfsv4.ko
modinfo nfsd
filename:      /lib/modules/3.10.0-327.13.1.el7.x86_64/kernel/fs/nfsd/nfsd.ko
modinfo nfs_acl
filename:      /lib/modules/3.10.0-327.13.1.el7.x86_64/kernel/fs/nfs_common/nfs_acl.ko
modinfo nfs_layout_flexfiles
filename:      /lib/modules/3.10.0-327.13.1.el7.x86_64/kernel/fs/nfs/flexfilelayout/nfs_layout_flexfiles.ko
modinfo nfs_layout_nfsv41_files
filename:      /lib/modules/3.10.0-327.13.1.el7.x86_64/kernel/fs/nfs/filelayout/nfs_layout_nfsv41_files.ko
```

5) Check linux kernel for ACL support (find *=Y option in accordance with the task)

```
uname -a
Linux rhel 3.10.0-327.13.1.el7.x86_64 #1 SMP Mon Feb 29 13:22:02 EST 2016 x86_64 x86_64 x86_64 GNU/Linux

grep -i acl /boot/config*
/boot/config-3.10.0-327.13.1.el7.x86_64:CONFIG_EXT4_FS_POSIX_ACL=y
/boot/config-3.10.0-327.13.1.el7.x86_64:CONFIG_XFS_POSIX_ACL=y
/boot/config-3.10.0-327.13.1.el7.x86_64:CONFIG_BTRFS_FS_POSIX_ACL=y
```

```
/boot/config-3.10.0-327.13.1.el7.x86_64:CONFIG_FS_POSIX_ACL=y
/boot/config-3.10.0-327.13.1.el7.x86_64:CONFIG_GENERIC_ACL=y
/boot/config-3.10.0-327.13.1.el7.x86_64:CONFIG_TMPFS_POSIX_ACL=y
/boot/config-3.10.0-327.13.1.el7.x86_64:CONFIG_NFS_V3_ACL=y
/boot/config-3.10.0-327.13.1.el7.x86_64:CONFIG_NFSD_V2_ACL=y
/boot/config-3.10.0-327.13.1.el7.x86_64:CONFIG_NFSD_V3_ACL=y
/boot/config-3.10.0-327.13.1.el7.x86_64:CONFIG_NFS_ACL_SUPPORT=m
/boot/config-3.10.0-327.13.1.el7.x86_64:CONFIG_CIFS_ACL=y
```

If there is N instead of Y, then it means linux kernel doesn't support ACL and need to be recompiled in accordance with the task.

6) Install NFS server and tools

```
yum install nfs-utils nfs4-acl-tools libnfsidmap
```

7) Check and enable NFS server services

```
systemctl list-unit-files | grep nfs
proc-fs-nfsd.mount                    static
var-lib-nfs-rpc_pipefs.mount          static
nfs-blkmap.service                    disabled
nfs-config.service                    static
nfs-idmap.service                     static
nfs-idmapd.service                    static
nfs-lock.service                      static
nfs-mountd.service                    static
nfs-rquotad.service                   disabled
nfs-secure-server.service             static
nfs-secure.service                    static
nfs-server.service                    disabled
nfs-utils.service                     static
nfs.service                           disabled
nfslock.service                       static
nfs-client.target                     enabled
```

```
systemctl enable nfs-server.service
systemctl enable nfs.service
systemctl start nfs-server.service
systemctl start nfs.service
```

8) Create NFS share and change permissions (the export filesystem) - a directories to share with client servers

```
mkdir -p /nfs
chmod a+rwxt /export
```

9) Share directories of NFS server for any (LAN, WAN, ...). Exports - NFS server export table.

```
vim /etc/exports
/nfs *(rw,fsid=0,nohide, no_root_squash, insecure,no_subtree_check,sync)
/nfs/tests *(rw,nohide, insecure,no_subtree_check,sync)
```

/nfs and **/nfs/tests -** shared directories

***** - users from any IP address of client machine are allowed to mount directories (means any client)

rw - allow both read and write requests on this NFS volume. The default is to disallow any request which changes the filesystem.

fsid=0 - export a directory over NFSv4. NFSv4 has a concept of a root of the overall exported filesystem. The export point exported with fsid=0 will be used as this root. The /nfs directory will be root for clients. For example, if you got /nfs/tests subdirectory, then client would see them as /tests directory. NFS needs to be able to identify each filesystem that it exports. For NFSv4, there is a distinguished filesystem which is the root of all exported filesystem. This is specified with fsid=root or fsid=0 both of which mean exactly the same thing. Only for "root" directory.

nohide - setting the nohide option on a filesystem causes it not to be hidden, and an appropriately authorised client will be able to move from the parent to that filesystem without noticing the change.

no_root_squash - turn off root squashing. This option is mainly useful for diskless clients. By default, any file request made by user root on the client machine is treated as by user nobody on the server. (Exactly which UID the request is mapped to depends on the UID of user "nobody" on the server, not the client.) If no_root_squash is selected, then root on the client machine will have the same level of access to the files on the system as root on the server.

insecure - option in this entry allows clients with NFS implementations that don't use a reserved port for NFS.

no_subtree_check - this option disables subtree checking, which has mild security implications, but can improve reliability in some circumstances. If a subdirectory of a filesystem is exported, but the whole filesystem isn't then whenever a NFS request arrives, the server must check not only that the accessed file is in the appropriate filesystem (which is easy) but also that it is in the exported tree (which is harder). This check is called the subtree_check.

sync - reply to requests only after the changes have been committed to stable storage (if async - improve performance, but at the cost that an unclean server restart (i.e. a crash) can cause data to be lost or corrupted). All changes to the according filesystem are immediately flushed to disk; the respective write operations are being waited for.

10) Restart NFS server services

```
systemctl restart nfs-server.service
systemctl restart nfs.service
```

11) Reexport all directories after modifying /etc/exports and display a list of shares files and export options on a NFS server

```
exportfs -r
exportfs -v
/nfs<world>(rw,wdelay,nohide,insecure,no_root_squash,no_subtree_check,fsid=0,sec=sys,rw,insecure,no_root_squash,no_all_squash)
/nfs/tests<world>(rw,wdelay,nohide,insecure,root_squash,no_subtree_check,fsid=0,sec=sys,rw,insecure,root_squash,no_all_squash)
```

12) Config or Disable firewall (firewalld or iptables services) on NFS server to allow client servers to access NFS shares. Open TCP port # 2049 which is used by NFSv4.

a) Config firewall

```
firewall-cmd --permanent --add-service nfs        *** need only one for remote mount via TCP port 2049
firewall-cmd --permanent --add-service rpc-bind
firewall-cmd --permanent --add-service mountd
firewall-cmd --reload

firewall-cmd --list-all
public (default, active)
interfaces: ens192
sources:
services: dhcpv6-client mountd nfs rpc-bind ssh
ports:
```

```
masquerade: no
forward-ports:
icmp-blocks:
rich rules:
```

```
cat /etc/services | grep mountd
mountd        20048/tcp            # NFS mount protocol
mountd        20048/udp            # NFS mount protocol
cat /etc/services | grep nfs
nfs          2049/tcp       nfsd shilp     # Network File System
nfs          2049/udp       nfsd shilp     # Network File System
nfs          2049/sctp      nfsd shilp     # Network File System
cat /etc/services | grep rpcbind
sunrpc        111/tcp       portmapper rpcbind     # RPC 4.0 portmapper TCP
sunrpc        111/udp       portmapper rpcbind     # RPC 4.0 portmapper UDP
```

b) Disable firewall

```
systemctl disable firewalld
systemctl stop firewalld
systemctl status firewalld
systemctl status firewalld
● firewalld.service - firewalld - dynamic firewall daemon
  Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; vendor preset: enabled)
  Active: inactive (dead)
```

13) Disable and SELinux

```
vim /etc/selinux/config

SELINUX=disable

***

sestatus

SELinux status:              disabled
```

14) Check all registered RPC programs (nfs, portmapper, mountd)

```
rpcinfo -p

  program vers proto   port  service

   100000    4   tcp    111  portmapper

   100000    4   udp    111  portmapper

   100024    1   udp  33569  status

   100024    1   tcp  41527  status

   100005    1   udp  20048  mountd

   100005    1   tcp  20048  mountd

   100003    4   tcp   2049  nfs

   100003    4   udp   2049  nfs

   100227    3   tcp   2049  nfs_acl

    100227    3   udp   2049  nfs_acl

   100021    1   udp  39271  nlockmgr

   100021    1   tcp  60402  nlockmgr
```

15) Set NFSv4 ACLs (add r/w permissions for user: he on directory: /nfs)

```
setfacl -R -m u:he:rwx /nfs
```

16) Check NFSv4 ACLs

```
getfacl -R /nfs

getfacl: Removing leading '/' from absolute path names

 # file: nfs

# owner: root

# group: root

user::rwx

user:he:rwx

group::r-x

mask::rwx

other::r-x

 # file: nfs/tests

# owner: root

# group: root

user::rwx

user:he:rwx

group::r-x

mask::rwx

other::r-x

 # file: nfs/tests/music_for_programming_00-manifesto.mp3

# owner: he

# group: he

user::rwx

user:he:rwx

group::rwx

mask::rwx

other::rwx

 # file: nfs/DDNTestTaskE1-E2-true.pdf

# owner: he

# group: he

user::rwx

user:he:rwx

group::rwx

mask::rwx

other::r-x
```

17) Install and settings addition software need for run testcases

a) rsh - remote shell access (need in order to receive commands from remote client)

Install rsh and rshd:

```
yum install rsh rsh-server
rpm -qa | grep rsh
rsh-server-0.17-76.el7_1.1.x86_64
rsh-0.17-76.el7_1.1.x86_64
```

Start rsh-server daemons:

```
systemctl enable rsh.socket
systemctl enable rlogin.socket
systemctl enable rexec.socket

systemctl start rsh.socket
systemctl start rlogin.socket
systemctl start rexec.socket

systemctl status rsh.socket
● rsh.socket - Remote Shell Facilities Activation Socket
  Loaded: loaded (/usr/lib/systemd/system/rsh.socket; enabled; vendor preset: disabled)
  Active: active (listening)
systemctl status rlogin.socket
● rlogin.socket - Remote Login Facilities Activation Socket
  Loaded: loaded (/usr/lib/systemd/system/rlogin.socket; enabled; vendor preset: disabled)
  Active: active (listening)
systemctl status rexec.socket
● rexec.socket - Remote Execution Facilities Activation Socket
  Loaded: loaded (/usr/lib/systemd/system/rexec.socket; enabled; vendor preset: disabled)
  Active: active (listening)
```

Configure rsh-server:

vim /root/.rhosts - allow the user root on the client fedora to log in as root on the target (server)

```
fedora root
```

vim /etc/securetty - enable external root user to execute the command (lists terminals from which root can log in)

```
rsh
rexec
rlogin
```

Client-side (hostname: fedora)

1) Modify hosts file in order to resolve IP from hostname

```
vim /etc/hosts
192.168.100.176 rhel
192.168.100.182 fedora
```

```
ping rhel
PING rhel (192.168.100.176) 56(84) bytes of data.
64 bytes from rhel (192.168.100.176): icmp_seq=1 ttl=64 time=0.594 ms
***
```

2) Disable firewall

```
systemctl disable firewalld
systemctl stop firewalld
systemctl status firewalld
systemctl status firewalld
● firewalld.service - firewalld - dynamic firewall daemon
  Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; vendor preset: enabled)
  Active: inactive (dead)
```

3) Disable and SELinux

```
vim /etc/selinux/config

SELINUX=disable

***

sestatus

SELinux status:                 disabled
```

4) Install nfs utils for client NFS service

```
yum install nfs-utils libnfsidmap
```

5) Enable and start NFS service

```
systemctl enable nfs-client.target
systemctl start nfs-client.target
```

6) Check nfs shares to the clients on NFS server

```
showmount -e 192.168.100.176

Export list for 192.168.100.176:

/nfs/tests *

/nfs       *
```

7) Mount the exported file system

```
mount -t nfs4 192.168.100.176:/ /nfs
```

Observe that only "/" is given instead of the actual exported path name

8) Check mounted NFS file system

```
mount | grep nfs
nfsd on /proc/fs/nfsd type nfsd (rw,relatime)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw,relatime)
192.168.100.176:/ on /nfs type nfs4
(rw,relatime,vers=4.1,rsize=524288,wsize=524288,namlen=255,hard,proto=tcp,timeo=600,retrans=2,sec=sys,clientaddr=192.168.100.182,local_lock=none,addr=192.168.100.176) *** ver 4.1
```

```
df -hT | grep nfs
192.168.100.176:/   nfs4     37G  5.5G   32G  15% /nfs
```

9) Mount NFS file system permanentrly in order to mount after reboot system

```
vim /etc/fstab
192.168.100.176:/   /nfs   nfs4 _netdev,auto  0  0
mount -a
```

10) Check all registered RPC programs

```
sdrpcinfo -p

 program vers proto   port  service

    100000    4   tcp    111  portmapper
    100000    4   udp    111  portmapper
```

11) Install and settings addition software need for run testcases

a) rsh - remote shell access (need in order to execute commands on remote server)
Install rsh and rshd:

```
yum install rsh rsh-server
rpm -qa | grep rsh
rsh-server-0.17-76.el7_1.1.x86_64
rsh-0.17-76.el7_1.1.x86_64
```

Client

Server

Test #1 <Test of ...>

Test #2 <Test of ...>

Test #3 <Test of ...>

Test #4 <Test of ...>

Test #5 <Test of ...>

Add

1)

ltp/include/mk/env_pre.mk

ltp/include/mk/env_pre.mk

To >>>>>

/cloud/Dropbox/sync/git/python/ltp/testcases/network/nfsv4/acl

2) file:///python/ltp-master/include/mk/env_pre.mk

```
find . -type d -exec chmod 755 {} \;
find . -type f -exec chmod 644 {} \;
```