SelfLinux-0.13.0

NFS - Network File System

Autor: Florian Frank (*florian.frank@pingos.org*)
Autor: Katja Socher (*katja@linuxfocus.org*)
Autor: Frédéric Raynal (*pappy@users.sourceforge.net*)
Formatierung: Florian Frank (*florian.frank@pingos.org*)

Ein Network File System (NFS) erlaubt das Verwalten von Dateien auf mehreren Computern innerhalb eines Netzwerkes so, als wären sie auf der lokalen Festplatte gespeichert. Dadurch ist es nicht nötig zu wissen, wo die Dateien physikalisch gespeichert sind, um auf sie zuzugreifen.

Inhaltsverzeichnis

1 Einführung

NFS erlaubt auf einfache Weise, Daten zwischen mehreren Computern zu teilen. Zum Beispiel muss sich ein Benutzer, der in einem Netzwerk angemeldet ist, nicht auch noch auf einem speziellen Computer anmelden; Über **NFS** hat er Zugriff auf sein home-Verzeichnis (man sagt, es ist **exportiert**) auf der Maschine, auf der er gerade arbeitet.

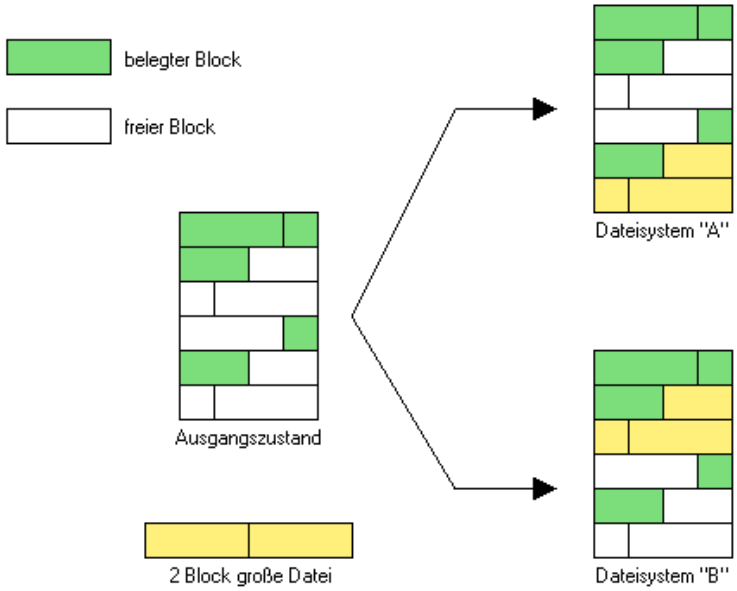
NFS ist **kein** sehr effizientes Protokoll und deshalb sehr langsam, wenn man es z. B. über eine Modemverbindung benutzt. Es wurde für ein lokales Netzwerk entwickelt und ist sehr flexibel. Es bietet eine Menge Möglichkeiten für Benutzer und Administratoren.

Man muss diesen Dienst allerdings mit Vorsicht verwalten. Jedem zu erlauben, Daten im Netz zu schreiben, wäre keine gute Herangehensweise. Einige essentielle Massnahmen reduzieren dieses Risiko. Sie werden später im Kapitel Potenzielle Sicherheitsprobleme beschrieben.

2 Allgemeine Darstellung von Dateisystemen

Bevor NFS beschrieben wird, sollte man den Ausdruck **Dateisystem** verstehen. Ein Dateisystem ist ein Weg, um Daten auf einem Medium zu speichern, der Weg, wie es organisiert und verwaltet wird. Es gibt viele davon, einige sind weit verbreitet, andere sind eher selten anzutreffen. Bekannte Dateisysteme sind etwa das New Technology FileSystem (NTFS von Windows), High Performance FileSystem (HPFS, OS/2), FAT 12/16/32, VFAT, Macintosh Hierarchical Filesystem (HFS), ISO 9660 (für CD-ROM), extended file systems (Ext, Ext2, Ext3), und viele andere.

Man kann jedes physikalische Medium für Daten (z. B. eine Festplatte) als Feld von kleinen Einheiten betrachten, die Informationen speichern: Man redet hier über **Blöcke**. Jedes Dateisystem verwaltet diese Blöcke auf verschiedene Weise. Zum Beispiel versuchen wir in der folgenden Abbildung eine Datei einzufügen, die zwei Blöcke benutzt. Auf der oberen Illustration wurde die Datei hinter den letzten belegten Block gesetzt, wobei leere Stellen am Anfang auftreten. Im unteren Teil des Bildes (ein anderes Dateisystem) wurde es an die erste freie Stelle gesetzt. Diese Politik hat Einfluss darauf, wie stark eine Platte fragmentiert wird. Einige Dateisysteme vermeiden Fragmentierung automatisch, während andere manuell defragmentiert werden müssen.



Das bekannteste Dateisystem unter Linux ist **ext2fs** (extended 2 file system). Jede Datei wird durch einen sogenannten **Inode** dargestellt. Inodes sind Datenstrukturen, die Informationen über die Objekte eines Dateisystemes speichern, wie z. B. Dateien oder Verzeichnisse. Verzeichnisse speichern die Dateiliste, und der Zugriff geschieht durch Operationen wie Lesen und Schreiben auf besondere Dateien.

Die Aufgabe eines **NFS Servers** ist es, den Clients die Inodes zu geben, auf die sie zugreifen möchten. Allerdings würde ein Client nur allein mit den Datei-Inodes nicht sehr gut arbeiten. Ein NFS Server ist eine zusätzliche Netzschicht, die es remote-Rechnern erlaubt, Inodes zu handhaben.

3 Das NFS Protokoll

Was man gewöhnlich **NFS** nennt, setzt sich aus vier verschiedenen Protokollen zusammen. Jedes hängt von **Remote Procedure Calls** (RPC) und `portmap` (auch `rpc.portmap` genannt) ab. Ein portmapper wandelt RPC Programmnummern in Portnummern um. Wenn ein RPC Server startet, teilt er `portmap` mit, welchen Port er benutzen wird, und die verwaltete RPC Programmnummer. Wenn ein Client eine RPC Abfrage an eine gegebene Programmnummer senden möchte, kontaktiert er zuerst den Server `portmap`, um die Portnummer zu bekommen, die ihm Zugang zu dem gewünschten Programm gibt. Dann adressiert er die RPC Pakete an den korrespondierenden Port.

Die vier Dienste, die es NFS erlauben zu arbeiten, sind:

3.1 Network File System (nfs)

Dieses Protokoll ist die Basis und erlaubt das Erzeugen von Dateien, Suchen, Lesen oder Schreiben. Dieses Protokoll verwaltet auch die Autentifizierung und die Dateistatistiken.

Der Daemon dieses Dienstes heisst `nfsd`.

3.2 Mount Daemon (mountd)

Dieser ist verantwortlich für das Mounten exportierter Systeme, um auf sie mit NFS zugreifen zu können. Der Server empfängt Anfragen wie `mount` und `umount` und muss auf diese Weise Informationen über exportierte Dateisysteme bewahren.

Der Daemon dieses Dienstes heisst `mountd`.

3.3 Network Status Monitor (nsm)

Es wird benutzt, um Netzwerknodes zu überwachen, um den Zustand einer Maschine (Client oder Server) zu kennen. Es informiert, z.B. über einen Neustart.

Der Daemon dieses Dienstes heisst `statd`.

3.4 Network Lock Manager (nlm)

Um Datenänderungen durch mehrere Clients zur gleichen Zeit zu vermeiden, verwaltet dieses Protokoll ein **Lock-System**. Es weiss, welche Dateien benutzt werden. Auf diese Weise ist es mit Hilfe des Nsm Protokolls möglich zu wissen, wann ein Client erneut startet. Nsm befreit alle Locks des Clients, bevor sie zurückgegeben werden.

Der Daemon dieses Dienstes heisst `lockd`.

3.5 Kernel NFS Daemon (knfsd)

Der Daemon `knfsd`, verfügbar ab **Kernelversion 2.4**, unterstützt direkt die nfs- und nlm-Protokolle. Andererseits werden `mountd` und `nsm` noch nicht unterstützt. Wenn der NFS Server installiert und gestartet wird, kann man mit dem folgenden Befehl verifizieren, dass alles läuft:

```
root@linux / # ps auxwww | egrep "nfs|mount|lock|stat"

root       220  0.0  0.0  1456  524 ?        S    Feb21   0:00
/sbin/rpc.statd
root      3944  0.0  0.0     0    0 ?        SW   Feb21   3:33 [nfsd]
root      3945  0.0  0.0     0    0 ?        SW   Feb21   0:00 [lockd]
root      3947  0.0  0.0     0    0 ?        SW   Feb21   3:42 [nfsd]
root      3948  0.0  0.0     0    0 ?        SW   Feb21   3:17 [nfsd]
root      3949  0.0  0.0     0    0 ?        SW   Feb21   3:00 [nfsd]
root      3950  0.0  0.0     0    0 ?        SW   Feb21   3:38 [nfsd]
root      3951  0.0  0.0     0    0 ?        SW   Feb21   3:28 [nfsd]
root      3952  0.0  0.0     0    0 ?        SW   Feb21   3:35 [nfsd]
root      3953  0.0  0.0     0    0 ?        SW   Feb21   4:32 [nfsd]
root      3956  0.0  0.1  1612  824 ?        S    Feb21   0:02
/usr/sbin/rpc.mountd
root     21528  0.0  0.0  1356  500 pts/0    S    12:58   0:00 egrep
nfs|mount|lock|stat
```

Momentan sind zwei NFS-Versionen verfügbar (Versionen 2 und 3 - sie werden **NFSv2** bzw. **NFSv3** genannt, um sie zu unterscheiden). **NFS4** befindet sich noch in der Entwicklungsphase, und ist im Kernel als **experimentell** gekennzeichnet.

Bei NFS geht es um eine Datenstruktur, die **file handle** genannt wird. Dies ist eine Bitserie, die auf eindeutige Weise erlaubt, jedes Dateisystemobjekt (wie eine Datei, aber nicht nur Dateien) zu identifizieren. Sie enthält z.B. den Datei-Inode, aber auch einen Eintrag, der das Gerät, auf dem sich die Datei sich befindet, enthält. Daher können wir NFS als ein Dateisystem betrachten, das in ein Dateisystem eingebettet ist.

4 Installation

4.1 Der Server

Das erste, das man tun muss, ist `portmap` zu starten, da dieses Protokoll von NFS benötigt wird.

```
root@linux / # /usr/bin/rpcinfo -p

rpcinfo: can't contact portmapper: RPC: Remote system error - Connection
refused
```

```
root@linux / # /sbin/portmap
```

Oder natürlich über das entsprechende Startskript der Distribution:

```
root@linux / # /etc/init.d/portmap start
```

```
root@linux / # /usr/bin/rpcinfo -p

       program vers proto   port
        100000    2   tcp    111  portmapper
        100000    2   udp    111  portmapper
```

Der Befehl `rpcinfo` zeigt die auf der Maschine laufenden RPC-Dienste, spezifiziert als das Argument (`-p` Option). Im ersten Fall sieht man, dass `portmap` noch nicht läuft: Man startet es (die meisten Linux-Distributionen enthalten Skripte, um dies beim Starten zu automatisieren). Danach wird erneut geprüft, ob es läuft. Ein anderer oft vorkommender Grund für eine negative Antwort auf `rpcinfo` ist, dass der portmapper nicht antworten darf aufgrund von Sicherheitsrestriktionen in den Dateien /etc/hosts.allow oder /etc/hosts.deny. In diesem Fall fügt man einen Eintrag zu der Datei /etc/hosts.allow hinzu.

/etc/hosts.allow

```
portmap: <hosts>
```

Vor dem Starten von NFS selbst muss es konfiguriert werden. Es gibt nur eine einzige Konfigurationsdatei, und die heisst /etc/exports. Jede Zeile zeigt einen Verzeichnisbaum, der exportiert werden soll, gefolgt von einer Liste von Clients, die auf ihn zugreifen dürfen. Es ist möglich, am Ende jedes Client-Namens Optionen hinzuzufügen. Die Manpage zu `exports` erklärt die Syntax für Client-Namen und Optionen.

Die akzeptierten Formen für Client-Namen sind:

* Rechnername
* Wildcards auf einen Domainnamen (z.B. : linux-*.mondomaine.fr)
* eine netgroup (Netzgruppe) (@group), wenn NIS benutzt wird

* eine IP-Adresse

/etc/exports

```
# /etc/exports: the access control list for filesystems which may be exported
# to NFS clients.  See exports(5).
/storage1/home          10.0.0.0/24(rw,async)
/storage1/system        10.0.0.0/24(rw,async)
/storage1/cdroms        10.0.0.0/24(rw,async)
/storage2/audio         10.0.0.0/24(rw,async)
/storage2/video         10.0.0.0/24(rw,async)
/storage2/backup        10.0.0.0/24(rw,async)
/storage2/transfer      10.0.0.0/24(rw,async)

/diskless/excelsior     excelsior(rw,async,no_root_squash)
/diskless/yorktown      yorktown(rw,async,no_root_squash)
/diskless/stargazer     stargazer(rw,async,no_root_squash)
/diskless/saratoga      saratoga(rw,async,no_root_squash)
```

Ohne im Detail alle verfügbaren `mount`-Optionen beschreiben zu wollen, hier dennoch eine Aufzählung der wichtigsten Optionen:

* `rw` (read write): Der Client kann im exportierten Sytem lesen und schreiben
* `ro` (read only): Der Client kann im exportierten System nur lesen
* `root_squash`: Es ist sicherer, wenn der root-Benutzer eines Client **nicht** mit Root-Erlaubnis schreiben darf. Um dies zu vermeiden, wird die UID/GID 0 (z. B. die von root) auf der Seite des Clients in den Benutzer **nobody** übersetzt. Diese Option ist standartmässig aktiv, kann aber mit `no_root_squash` abgeschaltet werden.
* `all_squash`: Alle Clients, die auf das oben exportierte System zugreifen, benutzen die UID/GID des Benutzers **nobody**
* `anonuid`, `anongid`: Der Benutzer **nobody** benutzt jetzt UID und GID definiert durch diese Optionen.

Jetzt müssen die `rpc.mountd` und `rpc.nfs` Daemons gestartet werden, um einen laufenden NFS-Server zu bekommen. Man sollte nochmals mit dem Befehl `rpcinfo` überprüfen, dass alles läuft. Man kann den Server sogar für die nsm- und nlm-Protokolle initialisieren (`rpc.statd` bzw. `rpc.lockd`). Sie sind aber nicht unbedingt notwendig, um einen NFS-Server laufen zu lassen. Allerdings sind sie hilfreich, wenn eine Maschine ausfällt, von selbst rebootet, etc...

Wenn man die Konfigurationsdatei /etc/exports ändert, muss man die betroffenen Daemons informieren, dass Änderungen gemacht wurden. Der Befehl `exportfs` übermittelt diese Information an die Server. Die Option `-r` synchronisiert die Datei /var/lib/nfs/etab mit der Datei /etc/exports. Die Option `-v` zeigt die exportierten Dateisysteme zusammen mit ihren Optionen an.

Nach dem Start enthalten die folgenden Dateien wichtige Informationen:

* /var/lib/nfs/rmtab: Jede Zeile zeigt den Namen des Clients und das Dateisystem, das von diesem Server importiert wurde.
* /var/lib/nfs/etab: Die Datei /etc/exports enthält nur eine Wunschliste. etab wird durch `exportfs` erzeugt. Es enthält auf jeder Zeile detaillierte Informationen über die Optionen, die benutzt werden, wenn ein Dateisystem zu einem einzelnen Client exportiert wird. Es ist die Referenzdatei, die von `rpc.mountd` benutzt wird, wenn dieser gestartet wird.
* /proc/fs/nfs/exports: enthält die Client-Liste, so wie sie der Kernel kennt.
* /var/lib/nfs/xtab: Wird benutzt für die Genauigkeit, wenn etab Clientennamen und Rechnergruppen mit **Wildcards** enthält. Diese Datei enthält nur explizite Rechnernamen.

Wenn ein Client auf ein Dateisystem zugreifen möchte, fängt er damit an, `mountd` zu fragen. Dieser sucht dann in etab, ob die Abfrage beantwortet werden kann. Er prüft ebenso, ob der Client zu dieser Abfrage berechtigt ist (hosts.{allow, deny}, Regeln der Firewall, ...). Der Kernel benutzt `exportfs` für die Überprüfung. Wenn in der Datei /var/lib/nfs/etab das exportierte System berechtigt ist, zu der Gruppe, zu der der Client gehört, exportiert zu werden, dann informiert `mountd` den Kernel, der daraufhin xtab mit dem neuen Rechnernamen aktualisiert.

4.2 Der Client

Hier gibt es normalerweise wenig zu tun. Der Zugriff auf ein Dateisystem, das von NFS exportiert wurde, wird direkt vom Kernel verwaltet. Er muss mit Unterstützung für NFS kompiliert worden sein. Die Datei /proc/filesystems listet alle Dateisysteme auf, die direkt vom Kernel unterstützt werden. Man muss dann nur dem Kernel sagen, dass man Zugriff auf ein von NFS exportiertes System haben möchte.

Der Befehl `mount` berechtigt zum Zugriff auf verschiedene Dateisysteme. Er informiert den Kernel, dass ein neues Dateisystem verfügbar ist, mit Angabe seines Typs, des Gerätes und seinem Mount-Punkt. Die Option `-t` kann dazu benutzt werden, um den Typ des benutzten Dateisystems zu spezifizieren. Für NFS schreibt man `-t nfs`.

`mount` hat seine eigenen Optionen für NFS. Zum Beispiel können die Optionen `rsize` und `wsize` dazu benutzt werden, um die Blockgrößen für Lesen und Schreiben zu ändern. Man kann NFS-spezifische Optionen mit allgemeineren Optionen wie `intr`, `noexec` oder `nosuid` verbinden. Die manpage zu `mount` listet alle diese Optionen auf.

Wir nehmen an, der Rechner **enterprise** ist ein NFS-Server und exportiert sein Verzeichnis /usr/local. Wenn man darauf vom Rechner **excelsior** aus zugreifen will, dann muss man nur das exportierte Verzeichnis von **enterprise** auf **excelsior** mounten:

```
root@linux / # mount -t nfs -o nosuid,hard,intr enterprise:/usr/local
/usr/local
```

Der Befehl zeigt an, dass man ein NFS Dateisystem mountet (`-t nfs`), mit den Optionen `nosuid`, `hard` und `intr`. Die beiden letzten Argumente sind die interessantesten. Das erste der beiden spezifiziert das zu mountende Gerät. Für NFS ist die Syntax anders als in der gewöhnlichen Befehlszeile von `mount`, wo man das Gerät und das Verzeichnis spezifiziert. Hier spezifiziert man **server:exported_directory** anstelle eines Gerätes.

Das letzte Argument zeigt die Stelle des Dateisystems auf der Client-Seite an. Hier teilen sich **excelsior** und **enterprise** einfach das Verzeichnis /usr/local. Dadurch kann vermieden werden, Programme mehr als einmal in /usr/local zu installieren. Um dies permanent zu machen, kann man dies auf **excelsior** in der Datei /etc/fstab spezifizieren. fstab enthält alle Dateisysteme, die beim Start gemounted werden müssen. Die Syntax für /etc/fstab lautet:

/etc/fstab

```
#    device           mount point file system  options          dump fsckorder
enterprise:/usr/local /usr/local  nfs          nosuid,hard,intr 0    0
```

Man sollte jedoch mit permanenten Einträgen vorsichtig sein. Man kann es nur benutzen, wenn der Server (**enterprise**) immer eingeschaltet ist oder vor dem Client (**excelsior**) eingeschaltet wird.

5 Potenzielle Sicherheitsprobleme

Ein grosses Problem mit NFS entsteht aufgrund der Tatsache, dass standardmässig ein **vertrauensvolles Verhältnis** zwischen Client und NFS-Server besteht. In dem Fall, dass der root-Account des Servers kompromittiert wird, wird auch der des Clients kompromittiert.

Ein Client darf nicht blind einem Server trauen und umgekehrt, deshalb muss man einschränkende Optionen spezifizieren, wenn der Befehl `mount` benutzt wird. Die erste wurde bereits erwähnt: `nosuid`. Es macht den Effekt des **SUID** und **SGID** Bits rückgängig. Das heisst, eine als root angemeldetet Person auf dem Server muss sich zuerst als Benutzer auf dem Client anmelden und wird erst dann root. Eine andere, restriktivere Option ist `noexec`. Diese verbietet das Ausführen von Programmen auf exportierten Dateisystemen. Diese Option ist nur auf Systemen sinnvoll, die nur Daten enthalten.

Auf der NFS-Serverseite kann man ebenso spezifizieren, dass dem root-Account des Client nicht vertraut werden soll. Man muss dies in /etc/exports mit der Option `root_squash` spezifizieren. Wenn ein Benutzer mit UID 0 (root) auf dem Client auf das Dateisystem, das vom Server exportiert wurde, zugreift, wird er für alle Abfragen zum Benutzer **nobody** umgewandelt. Diese Option ist standardmäßig unter Linux aktiv, kann aber mit der Option `no_root_squash` abgeschaltet werden. Eine Menge von UIDs kann spezifiziert werden, auf die diese Option angewendet werden soll. Es ist auch möglich, mit den Optionen `anonuid` und `anongid` Optionen den Zielbenutzer zu definieren, auf den die Anfrage umgewandelt werden soll.

Einige Aktionen sind allgemeiner und haben Auswirkungen auf den portmapper. Zum Beispiel kann man mit der folgenden Zeile in der Datei /etc/hosts.deny allen Rechner den Zugriff verbieten:

/etc/hosts.deny

```
# hosts.deny :
#             use the portmap
portmap: ALL
```

Die Datei /etc/hosts.allow stellt das Gegengewicht zu diesem strikten Verbot dar und erlaubt den Zugriff von den eingetragenen Maschinen.

Gute Firewall-Regeln tragen auch zu einem besseren Schutz bei. Es werden verschiedene Ports von den unterschiedlichen Diensten genutzt:

Service	Port	Protokoll(e)
portmap	111	upd / tcp
nfsd	2049	udp
mountd	variabel	udp / tcp

Weitere Informationen zur Sicherheit sind im Kapitel Grundlagen Sicherheit und Iptables zu finden.