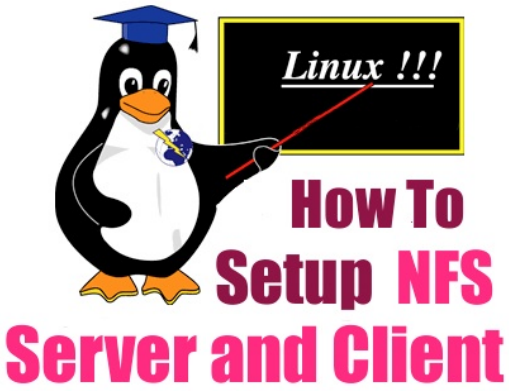
UNIVERSITY OF THE WEST of SCOTLAND

COMP09024 Unix System Administration

Laboratory 9: Network File System



NFS

Learning Outcomes

- **Setting up NFS for client and server**
- **Managing and tuning a shared network filesystem**

In this lab we will focus on the *Network File System* (NFS) protocol – a distributed file system protocol originally developed by *Sun Microsystems* in 1984. NFS follows a client/server model whereby client machines are able to access files over a network as if they are stored locally. Hence local workstations are able to conserve disk space because commonly used data can be stored on a single machine, and users need not establish separate home directories on every networked machine. As with many Unix applications, NFS can be set up simply by editing a few files and starting the appropriate daemons.

Installing NFS in a Debian distribution, using the apt-get tool

A NFS server needs only two programs, **`nfs-kernel-server`** and **`nfs-common`**, which we will download using **`apt-get`**. A client only needs the **`nfs-common`** facility. Nevertheless those packages are already installed in the virtual environment.

Note that both the server and the client will be the same virtual machine in this lab.

```
[root@UWS ~]# ifconfig
```

Note down your IP address, it should start with 10.x.x.x

Question

- **Q9.1)** Note the output line of the above command.

Now we can start to configure the client/server system.

NFS Server Configuration

First we will configure the server side.
Let us assume we want to share all files within a specific directory with our client. In the following example we are sharing **disk1**.

```
[root@UWS ~]# cd /media/disk1
```

Now, let us create a normal file in our shared directory filesystem **`/export/shared`** with:

```
[root@UWS ~]# echo 'Welcome to  IPSN' > file.dat
```

Now we enable full remote access with:

```
[root@UWS ~]# chmod 777 /media/disk1
```

Questions

- **Q9.2)** What permissions for user, group and world are given by the command **`chmod 777`** to the directory **`/media/disk1`**?
- **Q9.3)** What are the permission settings in octal representation for the newly created file **`file.dat`**?

All NFS exports from a server are controlled by the ASCII file **`/etc/exports`**, which the super-user is allowed to edit. Each line begins with the absolute path of a directory to be exported, followed by a space-separated list of allowed clients. For each entry of an allowed client, there is a list of specific control setting options within round parenthesis (). It is important not to leave any space between the last client specification character and the opening parenthesis, since spaces are interpreted as client separators.

So let us create or update the **`/etc/exports`** file on our system in an editor (e.g. gedit):

```
[root@UWS ~]# nano /etc/exports
```

Add an entry in **/etc/exports**, for the newly created shared directory file system. (Take care not to refer to file.dat itself, and not to type in the client machine name, instead of the IP-address.

General look of **/etc/exports**:

```
/media/disk1  IP(rw,no_root_squash,no_subtree_check)
```

Example for **/etc/exports**:

```
/media/disk1  10.5.187.25(rw,no_root_squash,no_subtree_check)
```

After editing, save the file and exit.

Question

- **Q9.4)** Explain the settings, **rw** and **no_root_squash**. Use the man pages and the Internet to find out. Why might **no_root_squash** be a dangerous setting for a shared filesystem?

A client can be specified either by name or IP address. Wildcards (*) are allowed in names, as are netmasks (e.g. /24) following IP addresses, but should usually be avoided for security reasons.

After editing **/etc/exports** we only need to tell the kernel about the fact, that we changed the file and want to export all (new) entries to our prospective clients. To make our changes effective please issue the command:

```
[root@UWS ~]# exportfs -a
```

Finally, we have to check that the **nfsd** daemon is up and running. Type the following command:

```
[root@UWS ~]# rpcinfo -p 10.5.187.187 | grep 2049
```

Do not forget to replace 10.5.187.187 by your IP address in the previous command.

Questions

- **Q9.5)** Which port is used by **nfs**?
- **Q9.6)** Which different protocol types are supported by **nfs** in your system?

As you should see **nfs** is already running. Try restart the daemon as follows:

```
[root@UWS ~]# /etc/init.d/S60nfs restart
```

Question

- **Q9.7)** Has the **nfs** service restarted?

And perform the following command to confirm that the server is running:

```
[root@UWS ~]# netstat -putan | grep <NFS_PORT>
```

Questions

- **Q9.8)** Does the **nfs** daemon connect to the same ports after restarting?
- **Q9.9)** Besides '**stop**' and '**start**' what other input options does the shell-script **/etc/init.d/S60nfs** accept? (Tip: locate the shell-script and do a **less**, **cat** or **tail** on the shell-script to see its contents. You will find all options towards the end of the shell-script, as arguments of the '**case**' selector. As the options are displayed within a case command, they will be indicated by a round bracket, immediately after their name: '*option_name)'* e.g. '**start**)').

That's it for the server side.

NFS Client Configuration

You should already have **nfs-common** installed on the client side, so we only need to create or dedicate an appropriate mount-point for the remote side. We do not need the remote filesystem itself to be presented in e.g. **/dev** or **/udev**, as this is already done within the remote machine! Moreover, the common mount command will also perform the remote mount! ow clever and smart is this? In order to indicate that this will be an imported filesystem let us create the following mount-point:

```
[root@UWS ~]# mkdir /mnt/imported_nfs
```

Remember to resolve the variable IPSN before creating the mount point! IPSN should be something like **'socomp-09'**. Also, bear in mind, that we just created a dedicated mount-point for this remote system. We could just as well have used an existing mount-point, e.g. **/home**.

After defining the mount-point on the client we just have to mount the remote filesystem:

```
[root@UWS ~]# mount IP:/media/disk1 /mnt/imported_nfs
```

To check whether the nfs-connection has worked we will try to set in to the directory:

```
[root@UWS ~]# cd /mnt/imported_nfs
```

and use the **more** command to check for the contents of the **file.dat**

```
[root@UWS ~]# more file.dat
```

Questions

- **Q9.10)** Note the contents of **`file.dat`** in your log-book
- **Q9.11)** Do you have permission to edit the file **`file.dat`** yourself from the remote system?
- **Q9.12)** In which system configuration file would you include a remote file system if you wanted it to be mounted at boot time?

That's all. To summarise we had to:

- Create a suitable mount point: **`/mnt/imported_nfs`**
- Tell the Kernel about it by mounting the remote file system on the server with the command: **`mount IPS:/export/shared /mnt/imported_nfs`**

Tuning NFS Performance

As a systems manager you may be called upon to tune NFS performance. The speed of NFS transactions is largely determined by how fast I/O requests are handled, which relies on having fast disks and a fast network. Processor speed and kernel optimisation only plays a minor role.

On the software side, by far the most effective step is to optimise the NFS block size. NFS transfers data in chunks. If the chunks are too small, a network node will spend more time processing packet headers than actually moving data. If the chunks are too large, a network node will move more bytes than necessary for a given amount of data.

To optimise the NFS block size, let us first measure the transfer time for various block size values. We will begin by determining transfer times for a 256 MB file:

```
[root@UWS ~]# umount /mnt/imported_nfs
```

```
[root@UWS ~]# mount IPS:/export/shared /mnt/imported_nfs -o rw,rsize=1024,wsize=1024
```

(on your system the IPS and IPSN variables should be replaced, as appropriate)

Please note, that you should not type a return between **`rw`**, and **`rsize=1024`**. Moreover, **`rw,rsize=1024,wsize=1024`** should be typed without spaces between any characters, as spaces are interpreted as file-system delimiters within this syntax. The above command is basically the same as the remote mount command before, but this time we set a defined read (**`rsize`**) and write size (**`wsize`**) for the chunks of data-blocks that will get transported via **`nfs`**. Both values of **`rsize`** and **`wsize`** are given in bytes.

Now we measure the transfer time on a block of just zero's, created by calling **`/dev/zero`**. We transfer the blocks with the help of the **`dd`** command. This command is used to do direct dumps from one device to a file or vice versa. The input file is given via '**`if=`**' and the output file is indicated with the '**`of=`**' assignment. The keywords **`bs`** and **`counts`** stand for blocksize and amount of blocks respectively.

```
[root@UWS ~]# time dd if=/dev/zero of=/mnt/imported_nfs/zeros.dat bs=1k count=2k
```

This should result in output resembling:

```
2048+0 records in
2048+0 records out

real    0m3.150s
user    0m0.038s
sys     0m0.389s
```

Questions

- **Q9.13)** Note the throughput of your Client/Server system for the nfs mount-options **`rsize=1024`** and **`wsize=1024`**, by performing the above command on your system and also noting the values for real, user and sys time
- **Q9.14)** Why is the '**`user`**' time so small? Please comment.
- **Q9.15)** Which activities will cause a non-zero system time (**`sys-time`**)?
- **Q9.16)** Why do you think **`rsize`** and **`wsize`** are optional parameters for nfs mount, but when mounting an internal device, these parameters are not provided?

Please check for the presence of the output file, **`zeros.dat`**, using the **`hexdump`** command.

```
[root@UWS ~]# od /mnt/imported_nfs/zeros.dat
```

Questions

- **Q9.17)** Note the output of the **`od`** command. Explain, referring to the man pages, the meaning of the asterisk that appears in the 2nd line of the output.
- **Q9.18)** Use **`od -vc /mnt/imported_nfs/zeros.dat`** as well and comment on the output. Please stop the output stream eventually by performing **`<CTRL-C>.`**

Here's a shell script to help you find an optimal blocksize for nfs communication… Edit and run the script for blocksizes of 1024, 2048, 4192, 8192, 16384 and 32768. In each case the shell script should time the operation three times. Before each individual run the script will unmount and remount the remote system. This is to defeat any caching of data that might otherwise be performed. Note the real time values output in seconds to two decimal places (e.g. 3.08s) and calculate an average for each blocksize. Record the results in your logbook in a table like the one below. You may find it helpful to use an Excel worksheet.

You should set yourself in the **`/root`** directory, before editing the script and remember to set execute permission and run the script by typing **`./scriptname`**

```
#!/bin/bash

echo -n 'Please enter blocksize: '
read blocksize
for (( c=1 ; c<=3 ; c++ ))
do
       umount /mnt/imported_nfs
       mount IP:/media/disk1 /mnt/imported_nfs -o rw,rsize=$blocksize,wsize=$blocksize
       time dd if=/dev/zero of=/mnt/imported_nfs/zeros.dat bs=1k count=2k
done
exit 0
```

R/Wsize=	1024	2048	4192	8192	16384	32768
Run:1 (real)						
Run:2 (real)						
Run:3 (real)						
Avg:						

Questions

- **Q9.19)** Comment on the behaviour of the real throughput time for increasing transfer block sizes, indicated by the sets of different **`rsize`** and **`wsize`** values. What is the optimal value for **`rsize, wsize`** for which the fastest real time throughput is achieved?
- **Q9.20)** Comment on the fact, that the system time (**`sys`**) stays nearly constant

Your optimal block sizes for both reading and writing will almost certainly exceed 1024 bytes. It may happen that your data do not indicate a clear optimum, but instead seem to approach an asymptote as block size is increased. In this case, you should pick the lowest block size that gets you close to the asymptote, rather than the highest available block size; evidence indicates that block sizes that are too large can cause problems.

– END OF LAB –