B4: Network filesystems

Preparation:

adding user with home directory

sudo adduser --home /home/TESTUSER1 testuser1

sudo adduser testuser2

changing user account

su testuser1

2. Configuring and testing NFS

2.1

#installing nfs server on lab1

sudo apt update

sudo apt install nfs-kernel-server

editing ‘/etc/exports’ file on lab1

‘/home 192.168.1.11(rw)

restarting nfs service

sudo systemctl restart nfs-server.service

sudo systemctl status nfs-kernel-server.service

installing nfs client on lab2

sudo apt install nfs-common

mounting the shared directory via nfs on lab2

sudo mount -t nfs 192.168.1.10:/home /mnt

2.2

Tunneling NFS Through SSH

One method of encrypting NFS traffic over a network is to use the port-forwarding capabilities of ssh.

```
# ssh root@192.168.0.42 -L 250:localhost:2049  -f sleep 60m
# ssh root@192.168.0.42 -L 251:localhost:32767 -f sleep 60m
```

The above command causes ssh on the client to take any request directed at the client's port 250 and forward it, first through sshd on the server, and then on to the server's port 2049. The second line causes a similar type of forwarding between requests to port 251 on the client and port 32767 on the server. The localhost is relative to the server; that is, the forwarding will be done to the server itself. The port could otherwise have been made to forward to any other machine, and the requests would

look to the outside world as if they were coming from the server. Thus, the requests will appear to NFSD on the server as if they are coming from the server itself. Note that in order to bind to a port below 1024 on the client, we have to run this command as root on the client. Doing this will be necessary if we have exported our filesystem with the default secure option.

3. Configuring and testing samba

unmounting the NFS share

sudo umount -t nfs 192.168.1.10:/home /mnt
umount “/mnt/home”

# installing samba server
sudo apt update
sudo apt install samba

# checking samba service
sudo systemctl status smbd

# adding testuser1 to the samba server
sudo smbpasswd -a testuser1

setting up samba client on lab2

sudo apt-get update
sudo apt-get install cifs-utils

# mounting TESTUSER1 folder on lab2
sudo mount -t cifs //lab1/homes/TESTUSER1 /mnt/samba-share/ -o username=testuser1

3.2
To solve this problem, we add remote user to the samba server and when the user tries to mount using sudo command, it asks for the samba user’s password.

4. Configuring and testing sshfs

4.1

# installing sshfs on lab2
sudo apt-get update
sudo apt-get install sshfs

mounting the remote directory or file
sshfs testuser1@192.168.1.10:/home/TESTUSER1 mnt/

unmounting the directory
umount mnt/

4.2

User-Level Mounting:
Users can mount remote directories without requiring administrative privileges. This is convenient for individual users who need to access their own or shared files on a remote server.

Secure Data Transfer:
Since SSHFS relies on the SSH protocol, data transfer between the local and remote machines is encrypted, providing a secure method for accessing and transferring files.

Secure Remote File Access:
When you need to access files on a remote server securely, SSHFS provides a secure and encrypted file system over the SSH protocol.

Additionally, it does not need to install any special server software like samba or nfs. It’s easy to setup and use.

4.3

- FUSE allows the implementation of file systems in user space rather than requiring kernel-space development.
- It provides isolation from kernel-space and if anything goes wrong with a FUSE file system, it will not affect or crash the whole system.

4.4

- Add computational overhead
- key management
- extra latency for encryption and decryption

5. Configuring and testing WebDAV

5.1

installing apache2 on lab1

Sudo apt update
sudo apt install apache2

sudo systemctl status apache2

enabling dav_fs module of apache2

sudo a2enmod dav
sudo a2enmod dav_fs

sudo systemctl restart apache2

creating and setting ownership of WebDAV directory

cd /*var*/www
sudo mkdir WebDAV
sudo chown www-data:1000 WebDAV

creating alias for *'/webdav' as '/var/www/WebDAV'*

cd '/etc/apache2/sites-availabe'
sudo nano 000-default.conf

Alias /webdav /var/www/WebDAV

<Directory /var/www/WebDAV>
    DAV On
</Directory>

sudo apachectl configtest # checking apache2 configuration
sudo systemctl restart apache2

 # installing elinks for checking webdav

sudo apt install elinks
elinks
http://localhost/webdav

sudo a2enmod auth_digest
sudo htdigest /var/www/WebDAV/users.password webdav testuser

 # testing with cadaver from lab2

sudo apt install cadaver
cadaver http://lab1/webdav
username: testuser
password: 1234

> ls # to list all the files
> edit newfile # to edit newfile
> get newfile # to download the newfile

 # mounting webdav to local filesystem

sudo apt-get install davfs2
sudo mount -t davfs http://localhost/webdav /mnt/webdav/
> username: testuser
> password: 1234

 # enabling version control

curl -X VERSION-CONTROL -u username:password -H "Depth: infinity"
http://webdav-server/path/to/resource

curl -X PROPFIND -u username:password -H "Depth: infinity"
http://webdav-server/path/to/resource

2.2.1. Creating a Version-Controlled Resource

In order to track the history of the content and dead properties of a versionable resource, a user can put the resource under version control with a VERSION-CONTROL request. A VERSION-CONTROL request performs three distinct operations:

1. It creates a new "version history resource". In basic versioning, a version history resource is not assigned a URL, and hence is not visible in the http scheme URL space. However, when the version-history feature (see Section 5) is supported, this changes, and each version history resource is assigned a new distinct and unique server-defined URL.
2. It creates a new "version resource" and adds it to the new version history resource. The body and dead properties of the new version resource are a copy of those of the versionable resource. The server assigns the new version resource a new distinct and unique URL.
3. It converts the versionable resource into a "version-controlled resource". The version-controlled resource continues to be identified by the same URL that identified it as a versionable resource. As part of this conversion, it adds a DAV:checked-in property, whose value contains the URL of the new version resource.

Note that a versionable resource and a version-controlled resource are not new types of resources (i.e. they introduce no new DAV:resourcetype), but rather they are any type of resource that supports the methods and live properties defined for them in this document, in addition to all the methods and live properties implied by their DAV:resourcetype. For example, a collection (whose DAV:resourcetype is DAV:collection) is a versionable resource if it supports the VERSION-CONTROL method, and is a version-controlled resource if it supports the version-controlled resource methods and live properties.

In the following example, foo.html is a versionable resource that is put under version control. After the VERSION-CONTROL request succeeds, there are two additional resources: a new version history resource and a new version resource in that version history. The versionable resource is converted into a version-controlled resource, whose DAV:checked-in property identifies the new version resource. The content and dead properties of a resource are represented by the symbol appearing inside the box for that resource (e.g., "S1" in the following example).

Normally, a resource is placed under version control with an explicit VERSION-CONTROL request. A server may automatically place every new versionable resource under version control. In this case, the resulting state on the server must be the same as if the client had explicitly applied a VERSION-CONTROL request to the versionable resource.

6. Raid 5

6.1

RAID: Redundant Array of Independent Disk, it provides a way of making data stores from multiple hard disk. It can be used for several purposes depends on the RAID level.

Parity: a single bit added to a block of data to detect error. Even parity and odd parity are two types of parity.

RAID 5: This level uses minimum three disks and has distribute parity information. It can tolerate a single disk failure. It can provide fault tolerance and redundancy. The parity information can be used to construct data from failed disk.

6.2

```
# adding extra disk to vm in vagrant
h.vm.disk :disk, size: "5GB", name: "disk1"
h.vm.disk :disk, size: "5GB", name: "disk2"
h.vm.disk :disk, size: "5GB", name: "disk3"

# creating the raid5 array
sudo mdadm --create --verbose /dev/md0 --level=5 --raid-devices=3 /dev/sda /dev/sdb /dev/sdc

# checking the raid status
cat /proc/mdstat

# attaching file system to the raid array
sudo mkfs.ext4 -F /dev/md0
```

```
# creating mounting point
sudo mkdir -p /mnt/md0

# mounting the raid array
sudo mount /dev/md0 /mnt/md0

# showing that raid5 is working
df -h -x devtmpfs -x tmpfs

# Permanently adding the array:
sudo mdadm --detail --scan | sudo tee -a /etc/mdadm/mdadm.conf
sudo update-initramfs -u
echo '/dev/md0 /mnt/md0 ext4 defaults 0 2' | sudo tee -a /etc/fstab
```

6.3

```
# mounting from lab2
sudo mount -t nfs 192.168.1.12:/mnt/md0 /mnt/raid-share/
```

7. Final question

Samba:

Use Cases:

- File and printer sharing between Windows and Linux/Unix systems.
- Providing a central file server for Windows-based networks.
- Media Streaming: Samba can serve media files (videos, music) to devices like smart TVs, media players, and game consoles.

Strengths:

- Cross-Platform Compatibility: Works well with Windows, Linux, and Unix systems.

Weaknesses:

- Complex Configuration: Setting up Samba can be intricate due to its various configuration options.
- Performance Overhead: Samba may introduce some overhead compared to native Windows file sharing.

NFS (Network File System):

Use Cases:

- Shared access to files and directories across Unix/Linux systems.
- Centralized storage for diskless clients.

Strengths:

- Efficient and lightweight protocol for Unix-based systems.

- Good performance for large-scale file sharing.

Weaknesses:

- Security: NFS lacks robust security features, making it vulnerable to unauthorized access.
- Limited Cross-Platform Support: Primarily used in Unix/Linux environments.

SSHFS (SSH File System):

Use Cases:

- Secure Remote File Access: SSHFS allows mounting remote directories securely over SSH.
- Backup and Synchronization: Users can mount remote directories for backup or synchronization.
- Access to Remote Servers: Developers and administrators can work directly on remote servers.

Strengths:

- Security: Utilizes SSH encryption for data transfer.
- Ease of Use: Simple setup and usage.

Weaknesses:

- Performance Overhead: SSH encryption can impact performance.
- Limited Scalability: Not ideal for large-scale deployments.

WebDAV (Web Distributed Authoring and Versioning):

Use Cases:

- Collaborative Editing: WebDAV enables collaborative document editing over HTTP/HTTPS.
- Remote File Management: Users can access and manage files remotely via web browsers or clients.

Strengths:

- Web-Based Access: Access files from any device with a web browser.
- Version Control: Supports versioning and locking.

Weaknesses:

- Performance: Slower than native file systems due to HTTP overhead.