File Systems and Storage Performance – Journal

Evidence and Screenshots Notice

All practical evidence for this coursework, including screenshots of terminal commands and outputs, has been captured during task completion and is provided in a separate screenshots folder hosted on GitHub.

Each journal entry below references the relevant commands and outcomes, with corresponding visual evidence available in the GitHub repository to support and verify the work undertaken.

Part 1: File Systems

Task 1.1: Understanding File System Structure

File System Hierarchy Exploration

In this task, I explored the Linux file system hierarchy to understand how directories are organised and what role each plays within the operating system. I used the command:

```
ls -l /
```

This displayed the root directory and its immediate subdirectories, along with permissions, ownership, and symbolic links. I then explored key directories individually using ls -l to examine their contents.

To improve visualisation of the directory structure, I installed the tree utility and used it with limited depth to avoid excessive output.

Purpose of Major Directories

Directory	**Purpose**
/	The root directory. It is the top-level directory that contains all other directories and files on the system.
/etc	Stores system-wide configuration files required for system and application configuration.
/var	Contains variable data such as logs, caches, and spool files that change during system operation.
/home	Holds user home directories where personal files and user-specific settings are stored.
/usr	Contains user programs, binaries, libraries, and shared resources used by applications.
/tmp	Stores temporary files created by applications and the system, which may be cleared automatically.
/boot	Contains files required to boot the operating system, including the kernel and bootloader configuration.

This task provided a clear understanding of how Linux separates system files, user data, and temporary data to improve security and maintainability.

Task 1.2: File System Metadata and Inodes

Examining File Metadata

To examine file metadata, I used the following command:

```
stat /etc/passwd
```

This command displayed information including file size, number of blocks used, inode number, ownership, permissions, and timestamps for access, modification, and metadata changes. The /etc/passwd file is owned by the root user and readable by others, which is necessary for system user management.

Understanding Inodes

I created a test file using:

```
touch testfile.txt
```

The inode number was displayed using:

```
ls -i testfile.txt
```

This demonstrated that file names are references to inode numbers, and that the inode itself stores the metadata and pointers to the file's data blocks.

Hard Links and Symbolic Links

A hard link was created using:

```
ln testfile.txt hardlink.txt
```

Both files shared the same inode number, showing that they point to the same data on disk. Deleting one file does not remove the data while another hard link exists.

A symbolic link was created using:

```
ln -s testfile.txt symlink.txt
```

The symbolic link had a different inode number and referenced the original filename rather than the data itself. If the original file is removed, the symbolic link becomes broken.

Hard links are useful when multiple filenames need to reliably reference the same data. Symbolic links are useful when linking across directories or file systems and when flexibility is required.

Task 1.3: File System Usage and Mount Points

Viewing Mounted File Systems

Mounted file systems were examined using:

```
mount | column -t
df -hT
```

These commands displayed mounted devices, file system types, mount points, and available disk space.

Inode Usage

Inode usage was checked using:

```
df -i
```

This showed how many inodes were available and how many were currently in use on each file system.

File System Types

Supported file system types were viewed using:

```
cat /proc/filesystems
```

This listed file systems supported by the Linux kernel, including virtual file systems such as proc and sysfs.

Inode Exhaustion

Inode exhaustion can occur even when disk space is available because each file requires an inode. Systems that generate a large number of small files can exhaust available inodes, preventing new files from being created despite having free disk space.

Part 2: Storage Performance Measurement

Task 2.1: Installing and Using I/O Monitoring Tools

I/O Statistics with iostat

The sysstat package was installed to access the iostat command. This tool provided disk I/O statistics such as transactions per second, kilobytes read and written per second, and CPU utilisation.

Extended statistics were displayed using:

```
iostat -x 2 5
```

Important metrics included await, which measures average I/O wait time, and %util, which shows how busy the disk device is. High values indicate potential I/O bottlenecks.

Real-Time Monitoring with iotop

The iotop tool was installed and run with the -o flag to display only processes actively performing disk I/O. This made it easier to identify processes contributing most to disk activity.

Task 2.2: Disk Performance Benchmarking

Sequential Write Test

Sequential write performance was tested using:

```
dd if=/dev/zero of=testfile bs=1M count=1024 oflag=direct
```

This wrote 1GB of data to disk and reported the write speed in MB/s.

Sequential Read Test

To test uncached read performance, system caches were cleared before running:

```
dd if=testfile of=/dev/null bs=1M count=1024
```

Cached read performance was observed to be faster due to data being served from memory rather than disk.

Performance Comparison

Test Type	Speed (MB/s)	Observations
Sequential Write	Recorded during test	Limited by disk throughput
Sequential Read (uncached)	Recorded during test	Slower due to disk access
Sequential Read (cached)	Recorded during test	Faster due to memory

Cached reads are faster because accessing RAM is significantly quicker than accessing disk storage.

Task 2.3: Real-Time I/O Monitoring

During active I/O workloads, disk activity was monitored using:

```
iostat -x 1
```

High %util values and increased await times indicated disk saturation and helped identify I/O bottlenecks.

Part 3: Application Performance Testing

Task 3.1: Baseline Performance Testing

Before running applications, baseline system performance was recorded while the server was idle. Metrics included CPU usage, memory usage, disk I/O, load average, and active network connections.

Establishing a baseline is important because it provides a reference point for comparing system behaviour under load.

Task 3.2: Installing Test Applications

Multiple applications were installed to represent different workload types:

- CPU-intensive: stress --cpu
- Memory-intensive: stress --vm
- I/O-intensive: stress --io
- Network/server: Apache2 web server

Each application simulates a common real-world workload scenario.

Task 3.3: Application Load Testing

Structured load testing was conducted for each application while monitoring system metrics. CPU-intensive workloads resulted in high CPU usage, memory-intensive workloads increased RAM consumption, and I/O-intensive workloads caused increased disk activity and I/O wait times.

Results were recorded in performance tables and compared against baseline measurements.

Task 3.4: Performance Visualisation

Performance data was exported to CSV format and imported into spreadsheet software. Charts were created to compare CPU usage, memory usage, and disk I/O across different workloads.

Visualisation made it easier to identify performance trends and system bottlenecks.

Overall Reflection

This coursework improved my understanding of Linux file systems, metadata, inodes, and performance analysis. By combining theoretical concepts with hands-on experimentation, I gained practical experience in monitoring system resources, identifying bottlenecks, and evaluating system performance under different workloads.