Department of Computer Engineering

Academic Term : January - May 2024

Class: B.E. (Computer) Semester VIII Division: B

Subject Name: Distributed Systems CSC 801

Name	**Riya Rajesh Patil**
Roll No.	**9282**
Experiment No.	**7**
Experiment Title	**Design and implement a distributed file system that allows multiple nodes to share and access files in a distributed environment.**

Aim - Design and implement a distributed file system that allows multiple nodes to share and access files in a distributed environment.

Postlab -

1. **What is NFS?**

   NFS stands for Network File System. It's a distributed file system protocol that allows clients on a network to access files and directories stored on remote servers as if they were local. NFS enables seamless sharing of files and resources across a network, providing a convenient and transparent way for users to access data stored on different machines.

   Key features of NFS include:

   1. Transparency: NFS abstracts the underlying network and file system details, allowing users to access remote files in a manner similar to local files.
   2. Scalability: NFS supports distributed file systems, enabling multiple clients to access shared resources concurrently, thus scaling to accommodate large numbers of users and data.
   3. Performance: It employs caching mechanisms to enhance performance by reducing the need for frequent network accesses for commonly accessed files.
   4. Security: NFS provides authentication and access control mechanisms to ensure secure access to files and prevent unauthorized users from accessing sensitive data.
   5. Interoperability: NFS is platform-independent and can be implemented on various operating systems, making it suitable for heterogeneous network environments.

2. **Mention different file accessing models & file caching schemes.**

   Different file accessing models include:

   1. Sequential Access: Data is read or written sequentially from the beginning to the end of a file. This model is suitable for tasks such as reading log files or processing data streams.

   2. Random Access: Allows direct access to any part of a file without the need to read or write data sequentially. Random access is essential for applications that require efficient access to specific file locations, such as databases.

   3. Stream-oriented Access: Data is processed in a continuous stream, and applications can read or write data in chunks or streams. This model is commonly used for reading or writing large volumes of data, such as multimedia files.

   4. Record-oriented Access: Files are organized into records, each containing multiple fields or attributes. Applications can access records individually for processing or manipulation, making this model suitable for database systems and structured data formats.

   File caching schemes include:

   1. Write-through caching: In this scheme, data is written both to the cache and the underlying storage simultaneously. It ensures that data consistency is maintained but may result in higher write latency due to the additional I/O operations.

   2. Write-back caching: Data is initially written only to the cache, and then asynchronously written to the underlying storage at a later time. Write-back caching can improve write performance by reducing the number of I/O operations but may introduce data consistency issues in case of system failures.

   3. Least Recently Used (LRU) caching: This scheme evicts the least recently used data from the cache when it reaches its capacity limit. LRU caching is based on the principle that recently accessed data is more likely to be accessed again in the near future.

   4. First-In-First-Out (FIFO) caching: Data is evicted from the cache in the same order it was originally stored. FIFO caching is simple to implement but may not always result in optimal cache performance, especially for workloads with varying access patterns.

   5. Random replacement caching: Data is evicted from the cache randomly when it reaches its capacity limit. While simple, this scheme may not be as efficient as LRU or FIFO caching in many scenarios.