Project Report

On

**NETWORK FILE SHARING SYSTEM**

Submitted in partial fulfillment of the requirements for the award of

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE & ENGINEERING**

(Artificial Intelligence & Machine Learning)

by

**Ms. D MEGANA LAXMI (22WH1A6603)**

**Ms. K VIJAYA RAJASREE (22WH1A6648)**

**Ms. B ANUSHA (22WH1A6655)**

**Ms. K PAVANI REDDY (22WH1A6664)**

**Under the esteemed guidance of**

**Ms. P Anusha**

**Assistant Professor, CSE(AI&ML)**



**Department of Computer Science & Engineering**

**(Artificial Intelligence & Machine Learning)**

**BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN**

**(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)**

**Accredited by NBA and NAAC with A Grade**

**Bachupally, Hyderabad – 500090**

2024-25

**Abstract :**

The Network File Sharing System is a client-server-based application enabling secure file sharing over a network. The server hosts files in a shared directory, dynamically listing available files for clients. Clients can connect, view, and download requested files using TCP/IP protocols. The system ensures efficient communication, data integrity, and scalability for multiple users. It is suitable for educational or organizational use, providing streamlined and accessible file-sharing functionality.The system supports concurrent client connections, ensuring seamless performance under load. Additionally, it can be extended to include user authentication and encrypted file transfers for enhanced security.

**Problem Statement :**

Develop a **Network File Sharing System** that addresses these challenges by providing a secure, scalable, and user-friendly platform. The system will enable efficient file sharing and collaboration, ensure robust security, and offer features like version control, role-based access control, cross-platform compatibility, and seamless backup and recovery options. This solution aims to enhance productivity, safeguard sensitive information, and foster effective collaboration in a connected work environment..

**Functional Requirements :**

· **File Storage and Retrieval**:

* The system should allow users to upload files to a shared network directory.
* Users should be able to download files from the shared network.
* The system should support various file formats (e.g., text, images, videos).

· **User Authentication and Authorization**:

* The system should require users to log in with a valid username and password.
* Role-based access control (RBAC) should be implemented, allowing different access levels (admin, user, guest).
* Admin users can assign or revoke permissions for different files or directories.

· **File Permissions**:

* Users should have read, write, and execute permissions on files or directories.
* The system must enforce permissions to prevent unauthorized access to files.
* Admin users should have the ability to modify permissions for shared files or directories.

· **File Sharing**:

* The system must allow users to share files or directories with other users.
* Shared files should be accessible based on the permissions granted by the user sharing them.

· **File Versioning**:

* The system should support version control for files to ensure that users can access previous versions of files.
* Users should be able to view a history of file versions and revert to a previous version if needed.

·

**Search Functionality**:

* The system should provide a search feature to find files by name, type, or other metadata (e.g., date modified).
* Users should be able to filter search results based on different criteria.

· **File Conflict Resolution**:

* The system should handle cases where two users try to modify the same file simultaneously (e.g., locking a file or alerting users about conflicts).

· **File Metadata Management**:

* The system should allow users to view and modify metadata (e.g., file name, size, date created, last modified).

· **Audit Logging**:

* The system must maintain logs of user activities, including file uploads, downloads, modifications, and access attempts.
* Logs should be available for review by system administrators.

· **Notification System**:

* Users should be notified when a file is shared with them or when their permissions are modified.
* Admins should receive alerts for any unusual activity or errors.

**Non - Functional Requirements :**

### ****Performance****

* **Response Time:** The system must ensure file access latency is below 5 milliseconds on a local area network (LAN).
* **Throughput:** Support a data transfer rate of at least 100 MB/s for simultaneous users.
* **Scalability:** Handle at least 500 concurrent file read/write operations without performance degradation.

**Reliability**

* **Availability:** Ensure 99.99% uptime for file access services, with failover mechanisms for hardware or software faults.
* **Fault Tolerance:** Recover from server or network failures within 30 seconds to maintain file access consistency.
* **Data Integrity:** Prevent data corruption during simultaneous read/write operations by implementing atomic operations.

**Security**

* **Access Control:** Implement role-based access control (RBAC) to restrict file operations (read/write/delete) by user or group permissions.
* **Encryption:** Use encryption for data in transit (e.g., TLS) and at rest to protect sensitive information.

**Usability**

* **User Interface:** Provide a user-friendly interface for configuration, monitoring, and troubleshooting the NFS.
* **Error Messages:** Deliver clear, actionable error messages for connection or file operation issues.

**Maintainability**

* **Configuration Management:** Allow dynamic updates to server configurations without requiring downtime.
* **Documentation:** Provide comprehensive documentation for setup, configuration, and troubleshooting.

**Portability**

* **Platform Independence:** Ensure the NFS can run on various server environments (e.g., virtual machines, cloud platforms, or physical servers).
* **Protocol Support:** Support industry-standard protocols like NFSv4 for broad compatibility with client systems.

**Scalability**

* **Horizontal Scalability:** Add more servers to the system to handle increased traffic without downtime.
* **Storage Expansion:** Support dynamic addition of storage devices or volumes without interrupting existing operations.

**Compliance**

* **Standards Adherence:** Comply with file-sharing and network communication standards (e.g., POSIX, RFC for NFS protocols).
* **Regulations:** Ensure compliance with relevant laws and guidelines (e.g., GDPR for data privacy, HIPAA for healthcare data).

**Efficiency**

* **Resource Utilization:** Minimize CPU and memory usage on both client and server systems during file operations.
* **Power Efficiency:** Optimize operations to reduce energy consumption in large-scale deployments.

**Interoperability**

* **Cross-Network Support:** Enable seamless file sharing across different network topologies (e.g., LAN, WAN, VPN).
* **Integration:** Allow integration with third-party applications, backup systems, and cloud services.

**Code :**

**1 . Server Code :** [server.c]

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <fcntl.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

void send\_file(int new\_sock, const char \*filename) {

    FILE \*file = fopen(filename, "rb");

    if (file == NULL) {

        perror("File not found");

        return;

    }

    char buffer[BUFFER\_SIZE];

    size\_t n;

    while ((n = fread(buffer, 1, sizeof(buffer), file)) > 0) {

        if (send(new\_sock, buffer, n, 0) == -1) {

            perror("Send failed");

            break;

        }

    }

    fclose(file);

    printf("File sent successfully\n");

}

int main() {

    int server\_sock, new\_sock;

    struct sockaddr\_in server\_addr, client\_addr;

    socklen\_t addr\_size;

    char filename[100];

    // Create socket

    server\_sock = socket(AF\_INET, SOCK\_STREAM, 0);

    if (server\_sock < 0) {

        perror("Socket creation failed");

        exit(1);

    }

    server\_addr.sin\_family = AF\_INET;

    server\_addr.sin\_port = PORT;

    server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

    // Bind the socket

    if (bind(server\_sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

        perror("Binding failed");

        exit(1);

    }

    // Listen for incoming connections

    if (listen(server\_sock, 10) == 0) {

        printf("Server is listening on port %d...\n", PORT);

    } else {

        perror("Listen failed");

        exit(1);

    }

    // Accept incoming connections

    addr\_size = sizeof(client\_addr);

    new\_sock = accept(server\_sock, (struct sockaddr \*)&client\_addr, &addr\_size);

    if (new\_sock < 0) {

        perror("Server accept failed");

        exit(1);

    }

    // Receive the filename from the client

    recv(new\_sock, filename, sizeof(filename), 0);

    printf("Client requested file: %s\n", filename);

    // Send the file

    send\_file(new\_sock, filename);

    close(new\_sock);

    close(server\_sock);

    return 0;

}

**2 . Client Code :** [client.c]

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <fcntl.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

void receive\_file(int sock, const char \*filename) {

    FILE \*file = fopen(filename, "wb");

    if (file == NULL) {

        perror("File creation failed");

        return;

    }

    char buffer[BUFFER\_SIZE];

    size\_t n;

    while ((n = recv(sock, buffer, sizeof(buffer), 0)) > 0) {

        fwrite(buffer, 1, n, file);

    }

    fclose(file);

    printf("File received and saved as %s\n", filename);

}

int main() {

    int sock;

    struct sockaddr\_in server\_addr;

    char filename[100];

    // Create socket

    sock = socket(AF\_INET, SOCK\_STREAM, 0);

    if (sock < 0) {

        perror("Socket creation failed");

        exit(1);

    }

    server\_addr.sin\_family = AF\_INET;

    server\_addr.sin\_port = PORT;

    server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    // Connect to the server

    if (connect(sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

        perror("Connection failed");

        exit(1);

    }

    // Get the filename to request

    printf("Enter the filename to request from the server: ");

    scanf("%s", filename);

    // Send the filename to the server

    send(sock, filename, sizeof(filename), 0);

    // Receive the file

    receive\_file(sock, filename);

    close(sock);

    return 0;

}

### **Compilation and Execution :**

1 . Compile the server and client programs :

    gcc server.c -o server

gcc client.c -o client

2 . Running the server.c file :

./server

3 . Running the client.c file:

./client

### **Output :**

Server Server side : Server started on port 8080

Client Side:   Enter 'LIST' to view available files or the filename to download: LIST

Available files:

file1.txt

file2.jpg

Enter 'LIST' to view available files or the filename to download: file1.txt

File 'file1.txt' downloaded successfully.

.