

**AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE, BENGALURU**

B. Tech in Artificial Intelligence and Data Science (AID)

Second Semester, Section F, Academic Year: 2023-24

Operating Systems (23AID213)

END-TERM PROJECT REPORT

on

**File Management System**

Presented by Group No. 17

**Leela Sabareesh (BL.EN.U4AID23037)**

**V. Vyvanth (BL.EN.U4AID23056)**

**Eda Manoj Krishna (BL.EN.U4AID23059)**

**Abstract:**

This project provides a sophisticated File Management System in Java that mimics key features of contemporary operating systems. The main objective is to provide a system to facilitate secure file operations like creation, reading, writing, and deletion while including memory management, disk space monitoring, and concurrency control. The system supports role-based access by distinguishing between administrators and users, with secure login and registration functionality. For handling concurrent access, Java semaphores are employed to avoid race conditions in file operations, providing thread safety and integrity.

The project simulates disk space usage, tracking total, used, and available memory dynamically to prevent overflow while writing or creating files. The system also provides a file-sharing mechanism among users and imposes a constraint of five open files per user to simulate real OS constraints. A basic menu-based interface allows users to interact with the system, perform operations, and monitor system status. The project is a general simulation of file management systems in real operating environments with a focus on concurrency control, memory management, and secure user interaction. The project provides a good learning experience of how primary file operations and system-level features are implemented and controlled under operating systems.

**Introduction:**

The most essential feature of any operating system is file management. It is responsible for keeping, storing, recovering, labeling, and protecting files on a disk. File systems in modern operating systems do far more than just storing data; they apply security, make use of the disk space, manage multiple users, and provide access by multiple users. With growing complexity and more demands by users, system programmers and programmers should understand the internal operation of file management systems.

The primary objective of this project is to design and implement a File Management System in Java that emulates the behavior of real operating systems to a large extent. It offers users safe interaction with files while imposing the correct constraints such as memory limits, file access restriction, and file sharing. The system includes concurrency, disk space monitoring, and safe access control in its design. The simulation not only helps to prove the logical operations involved in file management but also includes aspects of system programming such as multithreading and synchronization that are usually masked in high-level software programs.

Our system implements a role-based access model with two types of users: administrators and users. Login and registration assign each user a unique environment to work on his or her files. Administrators have more sophisticated capabilities such as reading all the files of the users and managing file access between accounts. Users can create, read, write, delete, open, close, and share files within their permission scope. Sharing files between users brings in real-world collaboration and demonstrates access control in a multi-user system. One of the subsystems' strengths lies in the disk space and memory management module. One virtual memory space is offered and dynamically controlled in terms of file operations. File creation or write operations are prevented from exceeding available memory, simulating actual operating system behavior closely. The system also imposes a per-user limit of five open files, akin to actual-world constraints prevalent in OS design.

Concurrency control is obtained by Java's semaphore mechanism that prevents race conditions when there are multiple threads reading, writing, or deleting files simultaneously. It is very useful in a multi-user system where operations may take place concurrently. Semaphores ensure mutual exclusion and protect shared resources from inconsistent access.

Briefly, this project seeks to create a solid simulation of a file system with increased complexity in concurrency control and memory management. It provides practical experience in efficiently and securely managing file operations and respecting system-level constraints, hence being a great learning tool to learn about file systems and operating system concepts.

**System overview:**

The File Management System created for this project simulates a multi-user setting where files can be safely created, accessed, shared, and managed. Object-oriented programming and multithreading are used in its Java implementation to mimic key file handling functions and system behaviours present in actual operating systems.

Administrator and Regular User are the two main user roles in the system. Permissions and access levels vary by role. To access the system, users need to register and log in. Through a menu-driven interface intended to improve user experience and reduce input errors, they can carry out a variety of file-related tasks after authenticating. In order to replicate a secure multi-user environment, user data, such as login credentials and file ownership, is securely managed.

The system's central component is a virtual file system, which uses internal data structures to manage files in memory instead of a physical filesystem. A unique Java class that contains metadata like the file name, content, owner, size, status (open or closed), and access permissions is used to represent each file. To avoid overflow or excessive use of memory, a simulated disc space of 10,000 bytes is allotted, and all file operations are checked against this limit. This enables users to experience realistic limitations, like running out of memory or being unable to create new files.

Each user is permitted to have up to five files open at once in order to mimic the constraints of real-world systems. By maintaining track of the number of open files per user session, this restriction is controlled and ensures that reading and writing can only take place when the file is open.

Java semaphores, which safeguard crucial code segments where shared resources are accessed, are used to accomplish concurrency control. For instance, semaphores guarantee mutual exclusion when several users try to read or write to a shared file in order to avoid data corruption or inconsistency. This feature mimics how operating systems handle concurrency in the real world.

The system's ability to facilitate file sharing is one of its noteworthy features. Users have the ability to share their files with others, and appropriate permission checks guarantee that only those who are authorised can access shared files. This simulates actual networked system situations and adds a layer of collaboration.

Administrators can view all files, examine user information, and control global file access, among other advanced features. Role-based access control is enforced by limiting regular users to their files and shared resources.

All things considered, this File Management System integrates several important ideas like memory management, user access control, and thread synchronisation to offer a thorough simulation of file handling in an operating system. It acts as a model for instruction and a starting point for comprehending how operating systems handle files in safe, multi-user, and resource-constrained environments.

**FEATURES AND FUNCTIONALITIES :**

The File Management System offers a rich set of features that closely emulate the real-time file operations and constraints enforced by modern operating systems. Below is a detailed breakdown of its core functionalities:

**1. User Authentication and Role Management**

* **Registration & Login**: New users can register with a username and password. Existing users can log in with valid credentials.
* **Role Assignment**: Users are categorized as **Administrators** or **Regular Users**. This role distinction governs the level of access and permissions throughout the system.

**2. File Operations**

* **Create File**: Users can create files by providing a file name and optional initial content. File size is checked against available disk space.
* **Read File**: Users can read content from a file, provided it is open and accessible.
* **Write File**: Allows appending or modifying file content. The system ensures that the write operation does not exceed available disk memory.
* **Delete File**: Users can delete their files or shared files if permitted.
* **Open/Close File**: Users must explicitly open a file before performing read/write operations. A limit of five open files per user is enforced to mimic real OS behavior.

**3. Memory and Disk Space Management**

* A virtual memory of **10,000 bytes** is maintained.
* Disk space usage is dynamically updated with each file operation.
* Users are notified if their action would exceed memory limits, encouraging efficient space usage.

**4. Concurrency Control with Semaphores**

* **Thread Safety**: Java semaphores are used to control access to critical file operations.
* **Race Condition Prevention**: Ensures that simultaneous file operations (like multiple users accessing or modifying a file) do not lead to data inconsistency or crashes.

**5. File Sharing Mechanism**

* Files can be shared with other registered users.
* Shared files are marked with proper metadata and access permissions.
* Only the owner or an admin can revoke or modify access rights.

**6. Admin-Specific Features**

* **Global File View**: Admins can view all user files and system memory usage.
* **User Oversight**: Admins can audit user activity and manage problematic accounts if needed.
* **Access Control Management**: Admins may restrict or allow file access on a system-wide level.

**7. User Interface**

* A **menu-driven console interface** guides users through operations in a step-by-step manner.
* Input validation and helpful prompts make the system user-friendly and error-resistant.

**8. File Metadata Tracking**

* Each file maintains metadata including:
  + File name
  + Owner
  + Content size
  + Access list
  + Open/closed status

**Architecture of the System / Module Description:**

1. **User Module**

Session tracking, registration, and authentication are handled by the User module. Credentials are saved in a secure format at the time of registration. A user session is created and the credentials are verified upon login. Permissions are determined by the roles that users are assigned, such as Regular User or Admin.

**2. File Module**

This is the core of the system. Each file is represented using a custom File class that includes:

* File name
* Owner
* Content
* Size
* Status (open/closed)
* Access list (for sharing)
* Creation and modification timestamps

Files are stored in a virtual storage structure, simulating disk memory. Each file operation such as create, read, write, delete, or share is routed through this module, which also checks for permission validity and available memory.

**3. Memory Management Module**

This module simulates a virtual disk with a memory limit (10,000 bytes). It keeps track of how much space is used and available. It ensures that:

* No file exceeds available space.
* Deletion frees memory.
* Users are warned of memory overflow.  
  This module interacts closely with the File Module.

**4. Concurrency Control Module**

Java Semaphores are used in this module to manage synchronisation. Semaphores guarantee mutual exclusion when several users try to view or edit the same file at the same time, avoiding race conditions or inconsistent data. To guarantee thread safety, a semaphore is used to guard each crucial part of the code, such as writing or opening files.

**5. Admin Module**

Accessible only to admin users, this module provides system-wide oversight features:

* Viewing all user accounts
* Viewing all file activities
* Managing permissions
* Monitoring disk usage  
  This enhances security and control in the multi-user environment.

6. **User Interface Module**

The system walks the user through a number of te.asks using a menu-driven command line interface. In addition to offering input prompts, validation, and error messages, it displays various menus according to the user's role.

**7. Implementation Details:**

The system walks the user through a number of te.Java's object-oriented features, support for multithreading, and strong class structures are used in the implementation of the file management system. The main elements of implementation are broken down in detail below:

1. **Programming Language:** Java

**2. Key Classes and Structures**

* User: Represents each system user. Stores username, password, role (admin/user), and a list of owned and shared files.
* File: Represents a file in the system. Stores content, name, size, owner, sharing list, and open/close status.
* FileManager: Handles creation, reading, writing, deletion, and sharing of files.
* MemoryManager: Simulates virtual memory (10,000 bytes), updates memory usage with file operations.
* AuthManager: Handles login, registration, and role-based permission checks.
* ConcurrencyManager: Uses Java’s Semaphore class to control access to shared resources.
* AdminPanel: Exposes methods for administrators to audit users and file activities.

**3. Data Handling**

* Files and user data are managed in-memory using Java collections like ArrayList, HashMap, and HashSet.
* For persistence (if required), the system can be extended to serialize user and file data to disk using Java I/O.

**Memory Simulation**

Although the system doesn’t use actual file storage, it simulates a **virtual memory** environment. A total memory capacity (10,000 bytes) is defined, and each file operation adjusts the used space accordingly. This mimics real-world OS constraints and teaches memory management concepts.

**9. Conclusion:**

The system walks the user through a number of te.The File Management System created for this project effectively replicates the core functions of an operating system's file handling. The system, which was developed with Java, includes essential features like file creation, reading, writing, deleting, and sharing while upholding stringent resource management and access control. The system's integration of concurrency control and memory management is one of its highlights; this improves the system's realism in simulating a real multi-user operating environment while also enhancing its robustness.

The system walks the user through a number of te.The system prioritised clarity and modularity throughout the development cycle. From memory tracking and concurrent access handling to user authentication, every module was made to function independently while integrating with other parts in a smooth manner. Future improvements, testing, and debugging are made simple with this modular approach.

The project successfully achieved its educational goalsJava programming and data structures

* User access management
* Memory allocation logic
* Java semaphores and thread synchronization
* Command-line based interfaces and menu-driven interaction models

**10. Future Work:**

The system walks the user through a number of te.There is a lot of room for future development to make the File Management System more scalable, user-friendly, and integrated with contemporary computing needs, even though it currently offers features like user authentication, file creation, deletion, reading, and writing.

Support for a variety of file types is one important area that needs work. The system is currently set up to handle files in plain text. Nevertheless, adding support for additional formats—like PDF, DOCX, CSV, and binary files—would increase the system's applicability in a wider range of fields. This would make the application appropriate for both personal and organisational file management by enabling users to efficiently store and manage various types of data.

Another enhancement would be the addition of a Command Line Interface (CLI). While the current system is GUI-based for ease of use, many technical users prefer command-line operations for efficiency and automation. A CLI would allow users to script file operations, perform batch processing, and integrate the file management system into developer toolchains and system workflows.