Mini Operating System

Session 2023 - 2027

Submitted by:

Muhammad Ali 2023-CS-128
Abdur Rahman Daniyal 2023-CS-111
Ghulam Mohiyudin 2023-CS-119

Supervised by:

Sir Nazeef ul Haq

Course:

Data Structures and Algorithms

Department of Computer Science

University of Engineering and Technology

Lahore Pakistan

Contents

Overview

The Mini Operating System is a comprehensive project designed to demonstrate advanced data structure and algorithmic concepts. It simulates core functionalities of an operating system, integrating features such as task scheduling, memory management, file handling, and network connectivity. The system employs multiple data structures like priority queues, linked lists, trees, and graphs to manage tasks, memory, files, and devices, while implementing sorting and traversal algorithms for efficiency. Additionally, it incorporates features like PDF report generation and network visualization to enhance usability and presentation.

Why This Project?

This project was developed to combine theoretical knowledge of data structures and algorithms (DSA) with practical application in solving real-world problems. By simulating an operating system, this project bridges the gap between foundational programming concepts and their advanced usage in system-level programming. It also aims to help students and developers better understand how operating systems function internally.

Objectives:

- Develop an interactive and user-friendly mini-operating system.
- Showcase task scheduling, memory management, and networking capabilities.
- Provide network visualization and reporting for improved analysis.
- Integrate theoretical DSA concepts into practical, real-world scenarios.
- Foster a deeper understanding of operating system internals and their design principles.

Target Audience

- **Students**: To gain hands-on experience with DSA and system-level programming. This project serves as a practical demonstration of how theoretical concepts can be applied to build real-world applications.

- **Educators**: To demonstrate practical applications of theoretical concepts and to provide a ready-to-use project for teaching advanced programming and systems design.
- **Developers**: To explore fundamental operating system functionalities, including task management, memory allocation, and networking.
- **Researchers**: To prototype enhancements in OS features or system simulations, and to experiment with algorithms in a controlled environment.

Tasks That Can Be Performed

Task Management

- **Store tasks**: Tasks are stored and executed based on their priority.
- **Sort executed tasks**: Merge Sort is used to display tasks in sorted order based on their priority.
- **View executed tasks**: A complete log of executed tasks is maintained for analysis and reporting purposes.

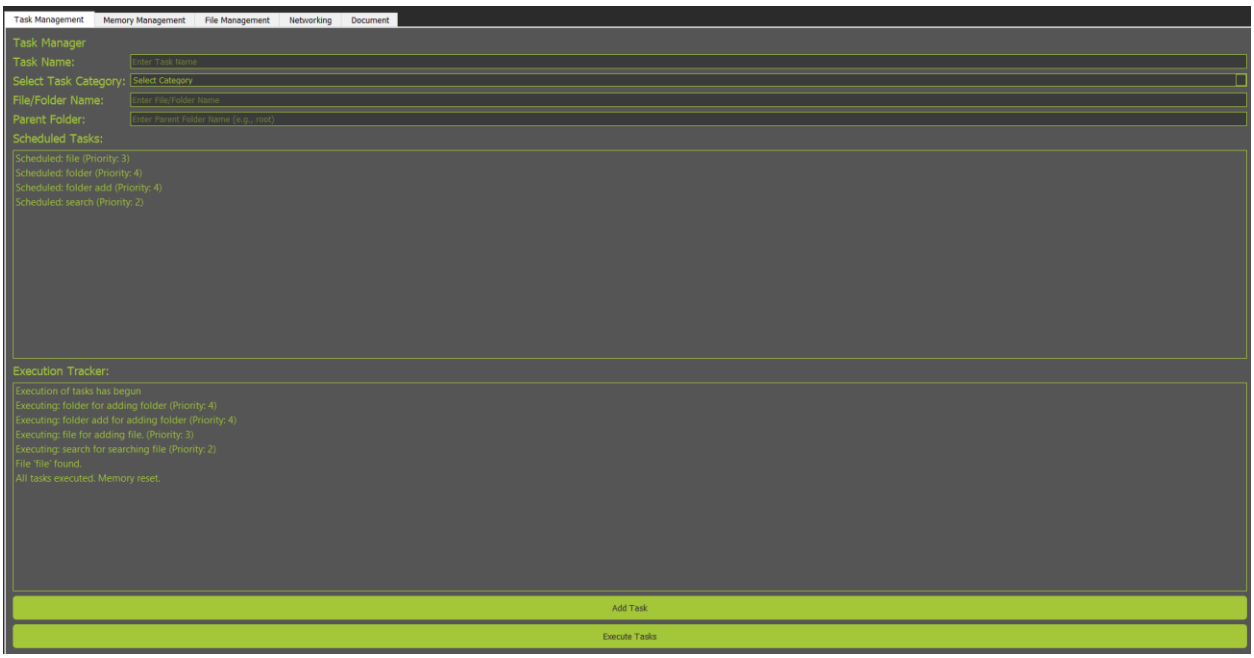

Figure 1 Task Management

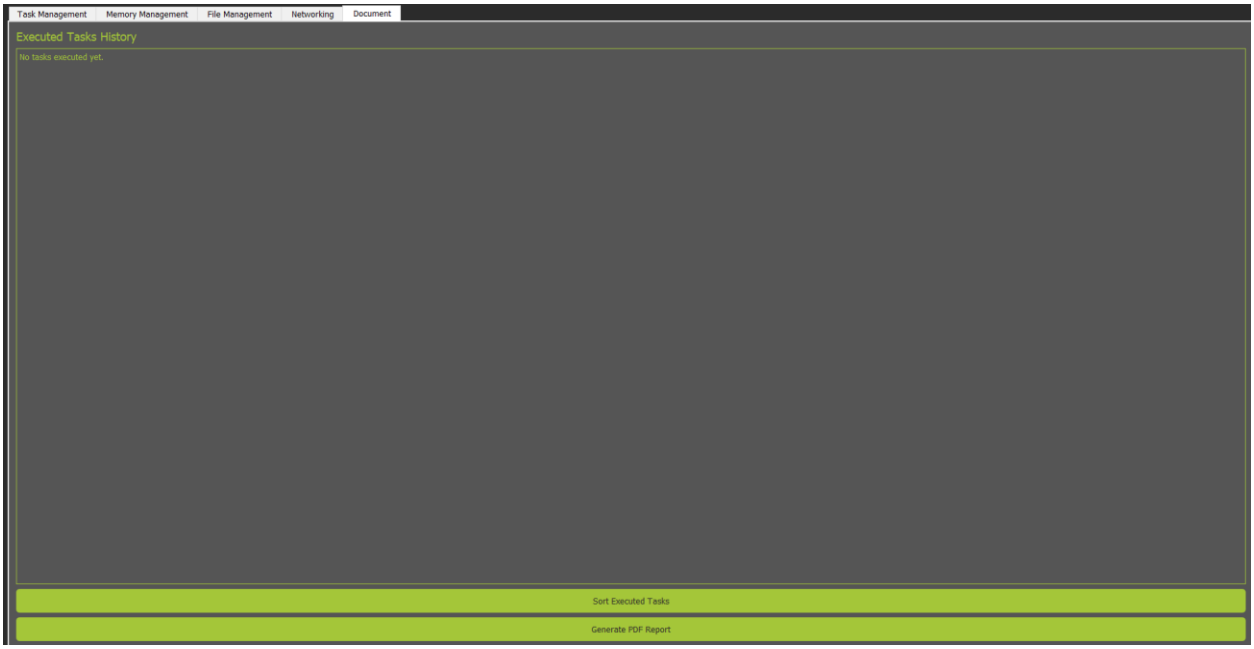

Figure 2 Task History

Memory Management

- **Simulate RAM**: Memory is managed using a linked list where memory blocks are allocated dynamically based on task priority.
- **Optimize memory usage**: Adjacent free blocks are merged automatically to prevent fragmentation and improve memory utilization.

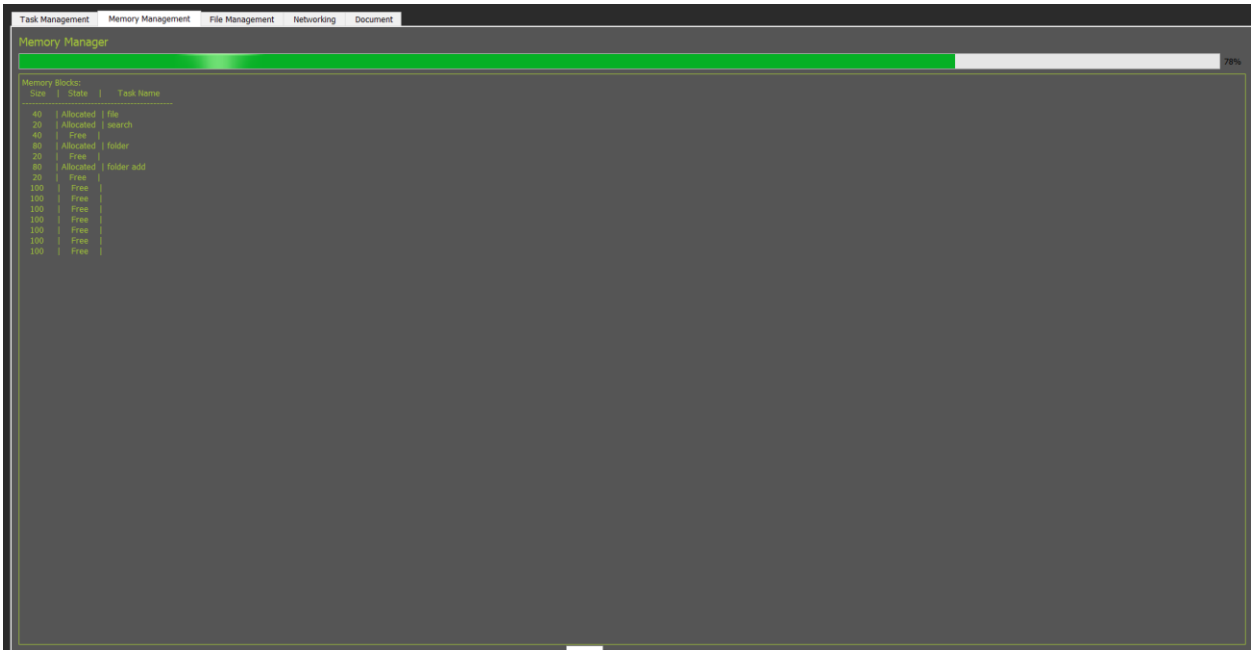

Figure 3 Memory Management

File System

- **Add file/folder**: Create new files or folders.
- **Remove file/folder**: Delete existing files or folders.
- **Search file/folder**: Locate a specific file or folder by name using efficient traversal algorithms.
- **Display file hierarchy**: Show the complete directory structure for better visualization and management.

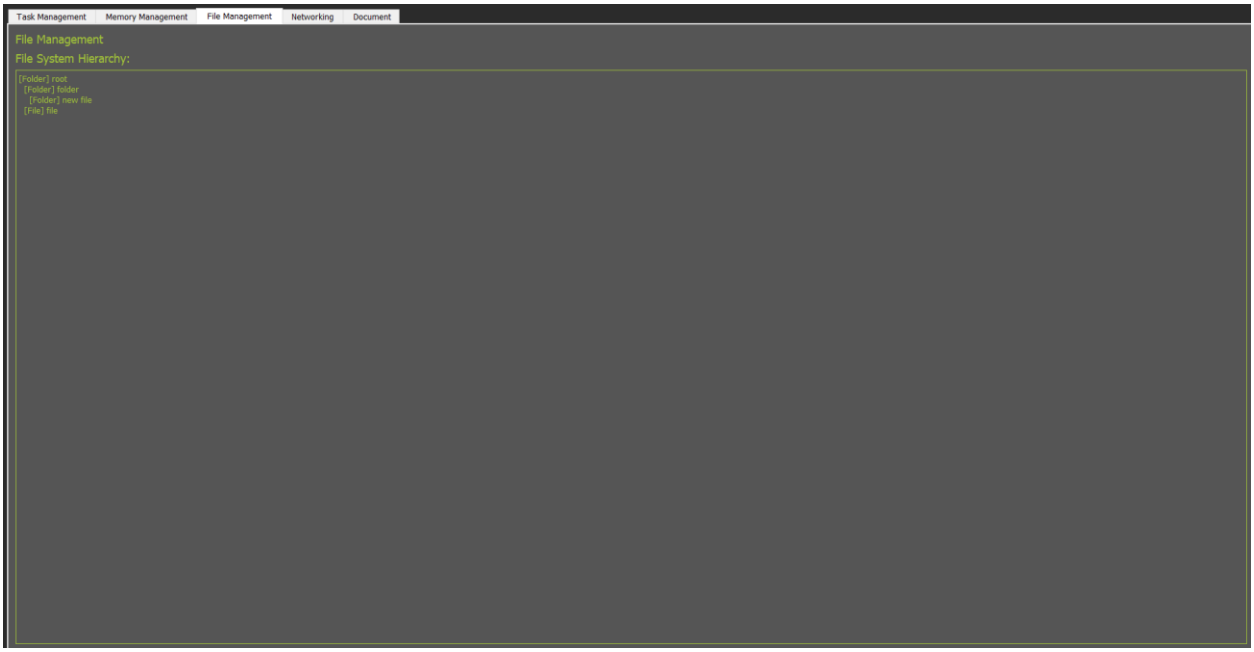

Figure 4 File System

Networking

- **Add device**: Add a new device to the network.
- **Remove device**: Remove an existing device from the network.
- **Add connection**: Establish a connection between devices.
- **Remove connection**: Disconnect devices.
- **Algorithms**: Perform operations like shortest path (Dijkstra's), breadth-first search (BFS), and depth-first search (DFS).
- **Network optimization**: Analyze network efficiency and suggest improvements based on connectivity and path lengths.

Figure 5 Network Management

Reporting and Visualization

- **Generate PDF Report**: A comprehensive PDF report of executed tasks.
- **Visualize Network**: Use matplotlib to create an intuitive network graph. Highlight key devices, critical paths, and connection bottlenecks.

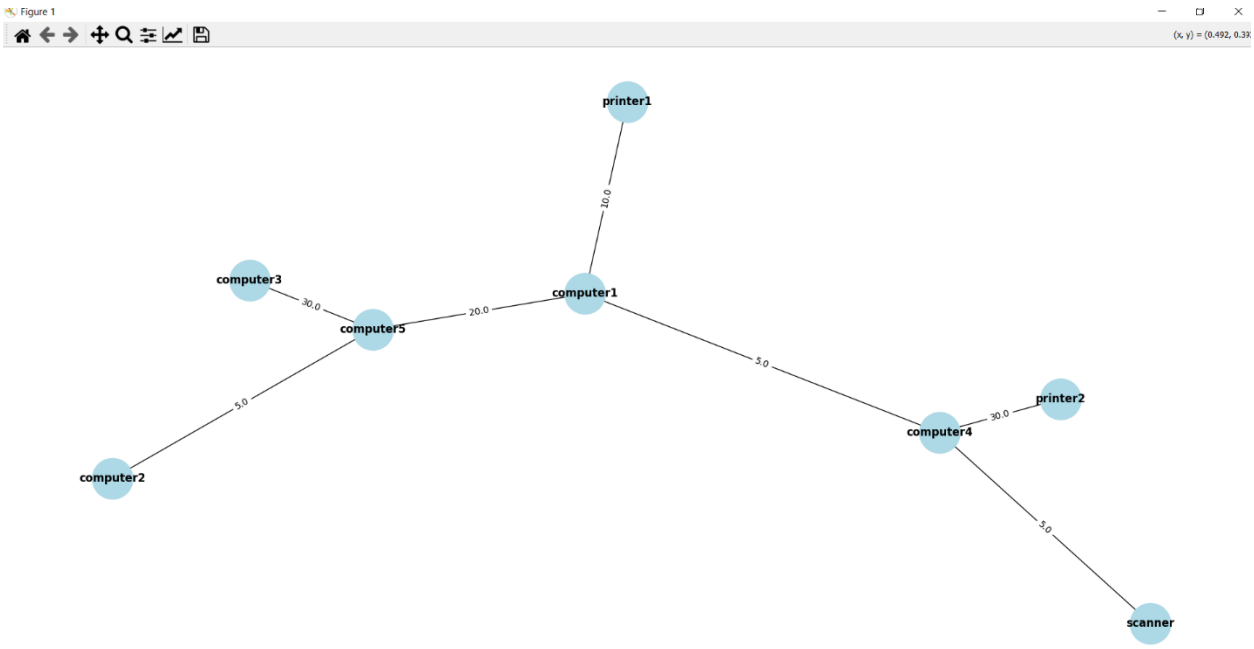

Figure 6 Network Visualizer

Data Structures Used

Priority Queue

- **Usage**: To store tasks and execute them based on their priority.
- **Implementation**: Min-heap or max-heap.
- **Key Operations**:
    - **Insertion**: Add tasks with associated priorities.
    - **Extraction**: Retrieve and remove the task with the highest priority.

Linked List

- **Usage**: To simulate RAM by dynamically allocating memory to tasks.
- **Implementation**: Singly or doubly linked list, where each node represents a memory block.

- **Key Operations**:
    - **Allocation**: Assign memory to tasks dynamically.
    - **Deallocation**: Free memory blocks after task completion.
    - **Merging**: Combine adjacent free blocks to optimize space.

Tree (File System)

- **Usage**: To manage files and folders hierarchically.
- **Implementation**: General tree structure where nodes represent files/folders.
- **Key Operations**:
    - **Insertion**: Add files or folders as nodes.
    - **Deletion**: Remove nodes and associated subtrees.
    - **Traversal**: Depth-first search to locate specific files or folders.

Graph (Networking)

- **Usage**: To represent devices as nodes and their connections as edges.
- **Implementation**: Adjacency list or adjacency matrix.
- **Key Operations**:
    - **Insertion**: Add nodes or edges dynamically.
    - **Deletion**: Remove nodes or edges.
    - **Traversal**: Perform BFS or DFS to explore connectivity.

Algorithms Used

- **Merge Sort**: Sort executed tasks by priority.
- **Dijkstra's Algorithm**: Find the shortest path between devices in the network.
- **BFS (Breadth-First Search)**: Traverse the network graph or find connected components.

- **DFS (Depth-First Search)**: Traverse the network graph or check connectivity.
- **Memory Optimization Algorithm**: Merge adjacent free blocks to reduce fragmentation.
- **File Search Algorithm**: Efficiently locate files or folders within a hierarchical structure.

Detailed Description of Components

Task Management

Priority Queue:

- Tasks are stored in a priority queue for execution.
- Higher priority tasks are executed before lower-priority tasks.
- Tasks can be dynamically added, updated, or removed.

Merge Sort:

- After execution, tasks are displayed in sorted order based on their priority.
- Ensures an organized and clear overview of executed tasks.

Memory Simulation

Linked List:

- Simulates RAM where each node represents a block of memory.
- Dynamic memory allocation is performed based on task requirements.
- Memory blocks are freed after task completion.
- Adjacent free blocks are merged to optimize memory usage.

Dynamic Allocation:

- Memory blocks are searched and allocated in real time.
- Tasks with no available memory wait until resources are freed.

File System

Tree Structure:

- Nodes represent files or folders with hierarchical parent-child relationships.
- Efficient management of large volumes of data.

Operations:

- **Add**: Append new nodes (files/folders) as children to a parent node.
- **Remove**: Delete nodes and associated subtrees.
- **Search**: Use depth-first traversal to locate specific nodes.
- **Scalability**: Supports future enhancements like metadata storage and file permissions.

Networking

Graph Representation:

- Devices are represented as nodes and connections as edges.
- Adjacency list ensures space efficiency.

Algorithms:

- **Dijkstra's Algorithm**: Finds the shortest path between devices.
- **BFS**: Traverses the graph to identify reachable nodes.
- **DFS**: Explores deeper connectivity and detects cycles.

Operations:

- **Add Device**: Introduce a new node to the graph.
- **Remove Device**: Delete an existing node and associated edges.
- **Add Connection**: Establish an edge between nodes.
- **Remove Connection**: Delete edges to sever device connectivity.

Reporting and Visualization

PDF Report:

- Summarizes executed tasks with details such as priorities and execution timelines.
- Useful for analysis and documentation.

Network Visualization:

- Uses matplotlib to create graphical representations of the network.
- Highlights specific devices and connections for better understanding.
- Facilitates troubleshooting and optimization.

Future Enhancements

Real-Time Task Scheduling:

- Implement algorithms like round-robin scheduling to handle tasks in real time.

Improved File System:

- Add support for metadata, version control, and file permissions.

Enhanced Networking Features:

- Introduce dynamic connection weights and support for real-time packet simulation.

User Interface:

- Develop a user-friendly GUI to simplify interaction with the system.

Performance Optimization:

- Optimize algorithms for better scalability and reduced computational overhead.

Cloud-Based Integration:

- Enable remote monitoring and task execution via cloud connectivity.

Energy Efficiency:

- Incorporate power management features to simulate real-world OS capabilities.

Advanced Reporting:

- Generate customizable reports with additional insights and analytics.

GitHub Repository:

The GitHub repo link of the project is given below:

https://github.com/aallliiii/operating-system-dsa.git

Conclusion

The Mini Operating System project encapsulates the essence of data structures and algorithms in a practical and engaging way. Its modular design ensures flexibility for future upgrades, making it a valuable tool for learning and development. By simulating key OS functionalities, it provides insights into task management, memory allocation, file handling, and networking—all crucial aspects of modern computing.