Project Report

1. Project Goals

Objective:
Develop a resource management and scheduling system for firefighting operations that efficiently allocates resources to multiple fire events based on their severity. The system should prevent resource deadlocks by implementing the Banker's Algorithm and prioritize resource allocation using Priority Scheduling.

2. Significance and Novelty of the Project

Background Information

In emergency situations like firefighting, efficient resource management is critical. Fire departments must allocate limited resources such as firefighting teams, vehicles, and water supplies to multiple fire events occurring simultaneously. Inefficient allocation can lead to resource deadlocks, where resources are either over-committed or under-utilized, potentially exacerbating the emergency.

Meaningfulness and Novelty

This project introduces a novel application of operating system algorithms—specifically, the Banker's Algorithm and Priority Scheduling—to the domain of firefighting resource management. By integrating these algorithms:

- **Deadlock Avoidance:** The Banker's Algorithm ensures that resource allocation does not lead to a deadlock situation, maintaining system safety.
- **Priority Handling:** Priority Scheduling based on fire severity ensures that more critical fires are addressed first, optimizing resource utilization.

This combination enhances the efficiency and reliability of resource management in firefighting operations, showcasing the practical utility of operating system concepts in real-world applications.

3. Installation and Usage Instructions

Prerequisites

- C compiler (e.g., GCC)
- Standard C libraries
- Terminal or command prompt access

Installation Steps

1. **Download the Source Code**
   Save the provided code into a file named fire_resource_management.c.
2. **Open Terminal**
   Navigate to the directory where fire_resource_management.c is saved.

Compile the Code

```
gcc -o fire_resource_management fire_resource_management.c
```

3. **Run the Program**

```
./fire_resource_management
```

4. **Usage Instructions**
1. **Enter the Number of Fire Events**
   When prompted, input the total number of fire events to manage.
2. **Input Fire Event Details**
   For each fire event:
   - **Location:** Enter the location name (string).
   - **Severity:** Enter a severity level between 1 and 10 (integer).
   - **Required Resources:** Input the required number of teams, vehicles, and water units.
   - **Maximum Resources:** Specify the maximum resources that might be needed for each resource type.
3. **Review Allocation Results**
   The program will:
   - Schedule the fire events based on severity.
   - Attempt to allocate resources using the Banker's Algorithm.

    - Inform you if resources were successfully allocated or if a deadlock was detected.
4. **Check Resource Logs**
   Resource usage is logged in resource_log.txt for auditing purposes.
5. **Program Termination**
   After processing all fire events, the program will release all resources and terminate.

4. Code Structure

Overview

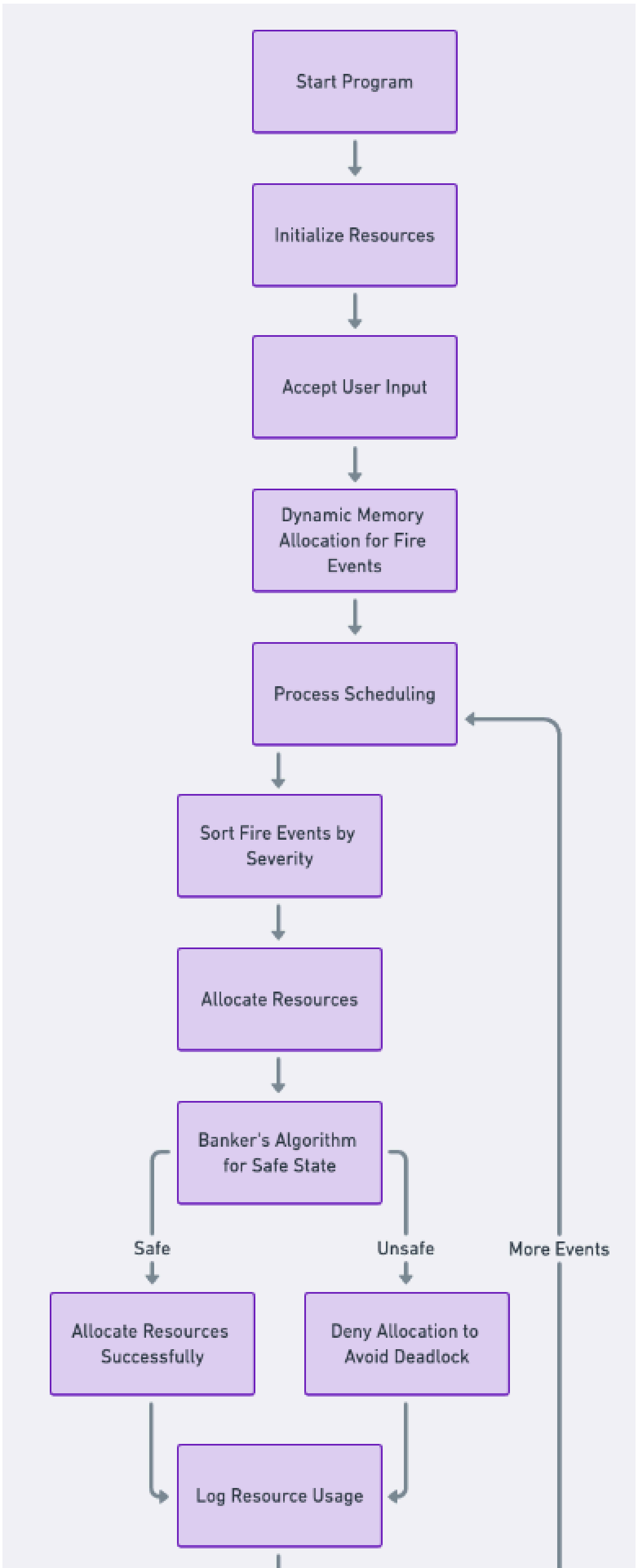
The program consists of several components:

1. **Data Structures**
    - Resource: Represents a resource type with total and allocated amounts.
    - FireEvent: Represents a fire event with required resources and status flags.
2. **Global Arrays**
    - Arrays for firefighting teams, vehicles, and water supplies.
    - Dynamic array for fire events.
3. **Functions**
    - **Initialization:**
        - initializeResources(): Initializes available resources.
    - **Scheduling and Allocation:**
        - processScheduling(): Prioritizes fire events based on severity.
        - allocateResources(): Allocates resources using the Banker's Algorithm.
        - isSafeState(): Checks for safe state to avoid deadlocks.
    - **Logging and Release:**
        - logResourceUsage(): Logs allocated resources to a file.
        - releaseResources(): Releases resources after fire event resolution.
4. **Main Function**
   Orchestrates the program flow by:
    - Accepting user input.
    - Initializing resources.

    - Scheduling and allocating resources to fire events.
    - Logging and releasing resources.
    - Cleaning up dynamically allocated memory.

Code Flow Explanation

1. **Initialization Phase**
    - The program starts by initializing resources and accepting the number of fire events from the user.
    - Dynamic memory is allocated for the fireEvents array.
2. **Input Phase**
    - The user inputs details for each fire event.
    - Required and maximum resources for each event are collected.
3. **Scheduling Phase**
    - Fire events are sorted based on severity using Priority Scheduling.
4. **Allocation Phase**
    - For each fire event:
        - The program attempts to allocate resources using the Banker's Algorithm.
        - If the allocation is safe, resources are allocated.
        - If not, the allocation is denied to prevent deadlocks.
5. **Logging and Release Phase**
    - Allocated resources are logged to resource_log.txt.
    - After the fire event is resolved, resources are released back to the system.
6. **Termination Phase**
    - After all events are processed, the program frees allocated memory and exits.


Start Program
Initialize Resources
Accept User Input
Dynamic Memory Allocation for Fire Events
Process Scheduling
Sort Fire Events by Severity
Allocate Resources
Banker's Algorithm for Safe State
Safe
Unsafe
More Events
Allocate Resources Successfully
Deny Allocation to Avoid Deadlock
Log Resource Usage

5. List of Functionalities and Verification Results

Functionalities

1. **Resource Initialization**
   - Initializes predefined resources for teams, vehicles, and water supplies.
2. **Dynamic Fire Event Handling**
   - Accepts a user-defined number of fire events with customizable details.
3. **Priority Scheduling**
   - Implements Priority Scheduling to sort fire events based on severity levels.
4. **Resource Allocation with Deadlock Avoidance**
   - Utilizes the Banker's Algorithm to allocate resources safely.
   - Prevents resource allocation that could lead to deadlocks.
5. **Resource Logging**
   - Logs resource usage details to resource_log.txt for record-keeping.
6. **Resource Release Mechanism**
   - Releases resources back to the pool after a fire event is resolved.

Verification Results

Test Case 1: Successful Allocation

- **Input:**
  - Fire events with resource requirements within available limits.
- **Result:**
  - Resources allocated successfully.
  - No deadlocks detected.
  - Resources released after use.

Test Case 2: Deadlock Prevention

- **Input:**
  - Fire event requesting more resources than available.
- **Result:**
  - Allocation denied.
  - Deadlock avoided.

  - System remains in a safe state.

Test Case 3: Priority Handling

- **Input:**
  - Multiple fire events with varying severities.
- **Result:**
  - Fire events processed in order of severity.
  - Higher severity fires receive resources first.

Test Case 4: Resource Logging Verification

- **Input:**
  - Multiple fire events with successful allocations.
- **Result:**
  - Resource usage correctly logged in resource_log.txt.

6. Showcasing the Achievement of Project Goals

Execution Results

Sample Scenario:

- **Number of Fire Events:** 3

Fire Event Details:

1. **Fire ID 0**
   - **Location:** "Downtown"
   - **Severity:** 8
   - **Required Resources:** 2 Teams, 1 Vehicle, 500 Water
   - **Maximum Resources:** 3 Teams, 1 Vehicle, 800 Water
2. **Fire ID 1**
   - **Location:** "Suburb"
   - **Severity:** 5
   - **Required Resources:** 1 Team, 1 Vehicle, 300 Water
   - **Maximum Resources:** 2 Teams, 1 Vehicle, 500 Water
3. **Fire ID 2**
   - **Location:** "Industrial Park"
   - **Severity:** 9

    - **Required Resources:** 3 Teams, 1 Vehicle, 700 Water
    - **Maximum Resources:** 4 Teams, 1 Vehicle, 1000 Water

Processing Sequence:

1. **Priority Scheduling**
    - Fire events are sorted based on severity:
        - Fire ID 2 (Severity 9)
        - Fire ID 0 (Severity 8)
        - Fire ID 1 (Severity 5)
2. **Resource Allocation**
    - **Fire ID 2:**
        - Allocation attempted.
        - Banker's Algorithm confirms safe state.
        - Resources allocated successfully.
        - Usage logged.
        - Resources released after resolution.
    - **Fire ID 0:**
        - Allocation attempted.
        - Safe state confirmed.
        - Resources allocated successfully.
        - Usage logged.
        - Resources released after resolution.
    - **Fire ID 1:**
        - Allocation attempted.
        - Safe state confirmed.
        - Resources allocated successfully.
        - Usage logged.
        - Resources released after resolution.

Achievement of Project Goals

- **Efficient Resource Management:**
    - Resources allocated without waste.
    - High-severity fires addressed first.
- **Deadlock Avoidance:**
    - Banker's Algorithm prevented unsafe allocations.
    - System remained in a safe state throughout.

- **Priority Handling:**
    - Severity-based scheduling ensured optimal response.
- **Verification through Logging:**
    - resource_log.txt confirmed correct resource usage.

7. Discussion and Conclusions

Project Issues and Limitations

1. **User Input Validation**
    - Limited error checking on user inputs.
    - Potential for invalid data leading to runtime errors.
2. **Static Resource Definitions**
    - Resources (teams, vehicles, water supplies) are predefined.
    - Not scalable for dynamic resource pools.
3. **Simplistic Modeling**
    - Does not account for resource attributes like location or capacity differences.
    - Assumes all resources are homogeneous.
4. **Single-threaded Execution**
    - Does not simulate real-time concurrent resource allocation.

Application of Course Learning

- **Operating Systems Concepts:**
    - Applied the Banker's Algorithm for deadlock avoidance.
    - Implemented Priority Scheduling for process management.
- **Data Structures:**
    - Used structs and arrays to model resources and fire events.
- **Dynamic Memory Management:**
    - Utilized dynamic allocation for the fire events array.
- **File I/O Operations:**
    - Employed file handling to log resource usage.
- **Algorithm Implementation:**
    - Translated theoretical algorithms into practical code.

Conclusions

This project successfully demonstrates the application of operating system algorithms to a real-world problem—firefighting resource management. By integrating the Banker's Algorithm and Priority Scheduling:

- **Efficiency Improved:**
  - Resources are allocated effectively, ensuring critical fires are prioritized.
- **Safety Ensured:**
  - Deadlocks are avoided, maintaining system stability.
- **Practical Utility Showcased:**
  - Highlights how theoretical concepts can solve practical issues.

Future Enhancements

- **Enhanced User Interface:**
  - Implement robust input validation and user-friendly prompts.
- **Dynamic Resource Management:**
  - Allow dynamic addition or removal of resources.
- **Concurrent Processing:**
  - Introduce multi-threading to simulate real-time resource allocation.
- **Advanced Scheduling Algorithms:**
  - Explore other scheduling methods like Round Robin or Shortest Job First.
- **Resource Attributes:**
  - Incorporate additional resource properties (e.g., team skill levels, vehicle capacities).

By addressing the limitations and expanding the functionalities, the project can evolve into a more comprehensive resource management system applicable to various emergency response scenarios.
