Banker's Algorithm Implementation

Documentation Overview

This documentation describes an implementation of the Banker's Algorithm for deadlock avoidance in C++. The solution consists of a single program that:

1. Reads resource allocation data from a file
2. Implements the Banker's Algorithm to determine if the system is in a safe state
3. If safe, outputs the safe sequence for process execution

The implementation uses a step-by-step approach to track resource allocation and determine a safe execution sequence.

System Architecture

Data Structures

```
// Process resource allocation matrices
vector<vector<int>> allocation;  // Currently allocated resources
vector<vector<int>> max;        // Maximum resources each process might request
vector<vector<int>> need;       // Resources still needed (max - allocation)

// Resource availability
vector<int> available;          // Currently available resources
vector<int> totalResources;     // Total resources in the system

// Safe sequence tracking
vector<int> safeSequence;       // Stores the safe execution order
vector<bool> finish;            // Tracks which processes have completed
```

Implementation Details

File Format

The input file (banker_data.txt) contains:

- Allocation matrix (5 rows × 3 columns)
- Maximum request matrix (5 rows × 3 columns)

- Available resources (1 row × 3 columns)

Key Functions

- main(): Sets up data structures, reads input file, and executes the algorithm
- isSafeState(): Core function that implements the Banker's Algorithm

Algorithm Explanation

The Banker's Algorithm works as follows:

1. Initialization:
    - Read allocation matrix, maximum matrix, and available resources
    - Calculate need matrix (need[i,j] = max[i,j] - allocation[i,j])
2. Safety Check Algorithm:
    - Initialize working vector with available resources
    - Create finish[] array initialized to false for all processes
3. Find Safe Sequence:
    - Find an unfinished process where need ≤ work
    - If found:
        - Add process to safe sequence
        - Mark process as finished
        - Release its resources (work = work + allocation)
        - Repeat until all processes complete
    - If not found at any point, system is not in a safe state
4. Check Result:
    - If all processes can complete, system is in a safe state
    - Output the safe sequence

Resource Management

The program properly manages system resources:

- Opens input file for reading resource data
- Allocates memory for matrices and vectors as needed

- Properly deallocates resources at program termination

Compilation and Execution

To compile the program:

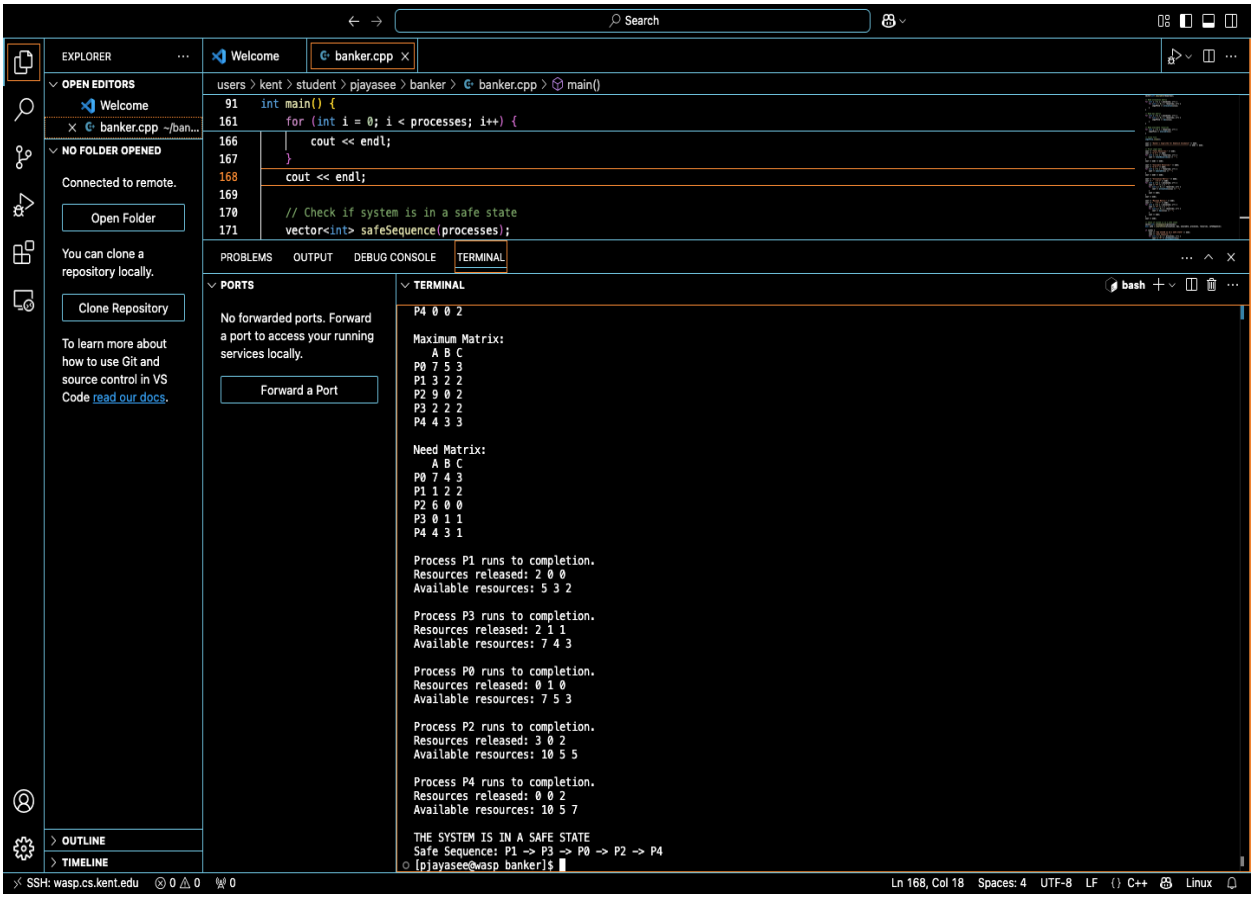
g++ banker.cpp -o banker

To run the program:

./banker

Sample Output

Attempt 1 and Success



Notes and Observations

- The system is in a safe state because there exists at least one sequence in which all processes can be executed to completion without deadlock.
- The algorithm prioritizes processes that can complete with the currently available resources.
- The step-by-step execution clearly shows how resources are released and reallocated during the safe sequence.
- The final available resources match the total resources in the system, confirming that all processes have completed and released their allocations.