Banker's Algorithm

A simple implementation of the Banker's Algorithm for deadlock avoidance in C++.

Requirements

- Linux/Unix operating system
- C++ compiler (g++ or clang++)

Environment

This project has been successfully tested on:

- Linux terminal
- University's Linux environment (WASP)

Compilation & Running

```
# Compile the program
g++ banker.cpp -o banker


# Run the program
./banker
```

Project Description

This implementation features:

- Analysis of resource allocation in a multi-process system
- Implementation of the Banker's Algorithm for deadlock avoidance
- Determination of safe sequence for process execution
- Detailed step-by-step resource allocation tracking
- Input data read from a customized file format

Sample Output

```
Banker's Algorithm for Deadlock Avoidance
Total Resources:
A B C
10 5 7
```

Available Resources:

A B C

3 3 2

Allocation Matrix:

A B C

P0 0 1 0

P1 2 0 0

P2 3 0 2

P3 2 1 1

P4 0 0 2

Maximum Matrix:

A B C

P0 7 5 3

P1 3 2 2

P2 9 0 2

P3 2 2 2

P4 4 3 3

Need Matrix:

A B C

P0 7 4 3

P1 1 2 2

P2 6 0 0

P3 0 1 1

P4 4 3 1

Process P1 runs to completion.

Resources released: 2 0 0

Available resources: 5 3 2

Process P3 runs to completion.

Resources released: 2 1 1

Available resources: 7 4 3

Process P0 runs to completion.

Resources released: 0 1 0

Available resources: 7 5 3

Process P2 runs to completion.

Resources released: 3 0 2

Available resources: 10 5 5

Process P4 runs to completion.

Resources released: 0 0 2

Available resources: 10 5 7

THE SYSTEM IS IN A SAFE STATE

Safe Sequence: P1 -> P3 -> P0 -> P2 -> P4

Project Structure

- banker.cpp: Main program implementing the Banker's Algorithm
- banker_data.txt: Input file containing allocation and resource data