Group Project 3: Banker's Algorithm

CECS 326 – Operating Systems

You should submit the required deliverable materials on Canvas by **11:59pm, March 30th (Sunday), 2025.**

1. Problem Description

The **Banker's Algorithm** is an important algorithm in Operating Systems used to avoid deadlocks in resource allocation. In this project, you will implement the Banker's Algorithm to simulate how a system allocates resources to processes in a safe manner while preventing deadlocks.

Requirements

- Basic understanding of resource allocation and deadlocks.
- Programming skills in a language of your choice (C, C++, Python, etc.).
- Familiarity with data structures like matrices and arrays.

1. System Definition
   - A system with n processes and m types of resources.
   - Each process requires a certain number of resource units.
   - Resources are allocated based on the maximum need of each process, and the system must decide whether to grant a resource request based on whether doing so will keep the system in a safe state.
2. Key Concepts
   - Available Vector: Represents the number of available units of each resource type.
   - Maximum Matrix: Represents the maximum number of units each process may request.
   - Allocation Matrix: Represents the current number of units allocated to each process.
   - Need Matrix: Represents the remaining resource needs of each process (i.e., Need = Max - Allocation).
3. Safety Algorithm
   - Before granting any resource requests, the system runs a Safety Algorithm to ensure that the system remains in a safe state.
   - A safe state ensures that there is at least one sequence of process execution that will not result in a deadlock.
4. Resource-Request Algorithm
   - If a process requests resources, the system checks if the request can be granted immediately by ensuring it does not exceed the maximum needs and that resources are available. It then simulates granting the request to check if the system remains in a safe state.

Sample output:

```
n = 5  # Number of processes
m = 3  # Number of resource types

# Available Vector (initially total resources available)
Available = [3, 3, 2]

# Maximum Matrix
Max = [
    [7, 5, 3],
    [3, 2, 2],
    [9, 0, 2],
    [2, 2, 2],
    [4, 3, 3]
]

# Allocation Matrix
Allocation = [
    [0, 1, 0],
    [2, 0, 0],
    [3, 0, 2],
    [2, 1, 1],
    [0, 0, 2]
]

# Need Matrix (Max - Allocation)
Need = [
    [7, 4, 3],
    [1, 2, 2],
    [6, 0, 0],
    [0, 1, 1],
    [4, 3, 1]
]
```

Test case 1: Run safe test:

```
System is in a safe state.
Safe Sequence: [1, 3, 4, 0, 2]
```

Test case 2:

```
# Process 1 requests resources [1, 0, 2]
request = [1, 0, 2]
request_resources(1, request)
```

Output:

```
System is in a safe state.
Safe Sequence: [1, 3, 4, 0, 2]
Resources allocated to process 1.
```

Test case 3:

```
# Process 4 requests resources [3, 3, 1]
request = [3, 3, 1]  # Process 4's resource request
```

```
    request_resources(4, request)
Output:

    Error: Not enough resources available.
```

2. The Required Deliverable Materials

(1) A README file, which describes how we can compile and run your code.
(2) Your source code, should submit in the required format.
(3) Your short report, which discusses the design of your program.
(4) A recorded video shows the output and runtime

3. Submission Requirements

You need to strictly follow the instructions listed below:

1) This is a **group project**, please submit a .zip/.rar file that contains all files, only one submission from one group.

2) Make a **video** to record your code execution and outputs. The video should present your name or time as identification (You are suggested to upload the video to YouTube and put the link into your report).

3) The submission should include your **source code** and **project report**. Do not submit your binary code. Project report should contain your groupmates name and ID.

4) Your code must **be able to compile**; otherwise, you will receive a grade of zero.

5) Your code should not produce anything else other than the required information in the output file.

7) If you code is **partially completed**, please explain the details in the report what has been completed and the status of the missing parts, we will grade it based on the entire performance.

8) Provide **sufficient comments** in your code to help the TA understand your code. This is important for you to get at least partial credit in case your submitted code does not work properly.

Grading criteria:

Details	Points
Have a README file shows how to compile and test your submission	5 pts
Submitted code has proper comments to show the design	15 pts
Screen a ***video*** to record code execution and outputs	10 pts

Have a **report** (pdf or word) file explains the details of your entire design	20 pts
Report contains clearly individual contributions of your group mates	5 pts
Code can be compiled and shows correct outputs	45 pts

4. Policies

1) Late submissions will be graded based on our policy discussed in the course syllabus.
2) Code-level discussion is **prohibited**. We will use anti-plagiarism tools to detect violations of this policy.