Assignment 08
CS341: Operating System Lab

---

1. Question: Write a C program to simulate deadlock avoidance using a Resource Allocation Graph (RAG). The program should check for the presence of a circular wait, which is an indication of potential deadlock. If a circular wait exists, the program should declare that the system is in deadlock; otherwise, it should report that the system is deadlock-free.

Assumptions:

1. There are `N` processes and `M` resources in the system.
2. Each process may request multiple instances of different resources.
3. A process can either be in a waiting state (requesting resources) or holding resources.
4. The system uses a directed graph where:
    - Each node represents either a process or a resource.
    - A directed edge from a process to a resource indicates that the process is waiting for the resource.
    - A directed edge from a resource to a process indicates that the resource is currently allocated to that process.

Instructions:

1. The program should accept input for the number of processes and resources.
2. Create and maintain a Resource Allocation Graph (RAG) using adjacency matrices.
3. The program should allow the user to input the current state of the system (i.e., which processes are holding or waiting for which resources).
4. Use **depth-first search (DFS)** or **cycle detection algorithms** to check if there is a circular wait (a cycle in the graph).
5. If a cycle exists in the graph, the system is in a deadlock state.
6. Output the result indicating whether a deadlock is detected or not.

Inputs:

Enter number of processes: 3
Enter number of resources: 3

Enter resource allocation (1 if process holds the resource, 0 otherwise):
Process 1: 1 0 0
Process 2: 0 1 0
Process 3: 0 0 1

Enter resource request (1 if process is requesting the resource, 0 otherwise):
Process 1: 0 1 0
Process 2: 0 0 1
Process 3: 1 0 0

Outputs:

Deadlock detected: Process 1 -> Resource 2 -> Process 2 -> Resource 3 -> Process 3 -> Resource 1 -> Process 1

2. Question: Given three resources A,B,C and total availability of all the resources x,y,z respectively. Find a valid sequence for which system does not give deadlock. if no such sequence exists, return -1.

Input Description:

The first line of the input contains one integer t (1≤t≤10^4) — the number of test cases.
Each test case consists of following n+2 lines:

1. First line contains one integer n (1≤n≤4 · 10^5) — the number of processes
2. Second line contains three integers x,y,z (1<=x,y,z<=10^5) where x,y,z are the amount of total resources available for A,B,C.
3. Next n lines contain x1, x2, x3, y1, y2, y3 (1<=xi, yi<=10^5) where (x1, x2, x3) are currently allocated resources to process pi and (y1, y2, y3) are max total requirement of the process pi for the resources A, B, C.

Inputs:

1
5
3 3 2
0 1 0 7 5 3
2 0 0 3 2 2
3 0 2 9 0 2
2 1 1 2 2 2
0 0 2 4 3 3

Outputs:

Following is the SAFE Sequence
P1 -> P3 -> P4 -> P0 -> P2

3. Question: Write a C program that demonstrates a deadlock situation using two threads. Each thread should attempt to lock two shared resources, but the locking sequence should be designed in such a way that both threads get into a deadlock state.

Assumptions:

1. Resources: There are two shared resources (e.g., resource A and resource B) that both threads need to access.
2. Threads: There are two threads (e.g., thread 1 and thread 2). Each thread tries to acquire both resources in a different order, leading to a deadlock.
3. Mutex Locks: Each resource is protected by a mutex lock to prevent race conditions when accessed by multiple threads.
4. Deadlock Condition:
    - Thread 1 locks resource A, then waits for resource B (held by thread 2).
    - Thread 2 locks resource B, then waits for resource A (held by thread 1).
    - Neither thread can proceed, resulting in a deadlock.

Example Scenario:

- Thread 1 locks resource A and tries to lock resource B.
- Thread 2 locks resource B and tries to lock resource A.
- Both threads end up waiting on each other, causing a deadlock.