DIVYANSHU MAKHARIA
221070036

# ASSIGNMENT 5

## AIM:

Implement Bankers algorithm for Deadlock avoidance.

## THEORY:

- The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for the predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.
- Deadlocks occur when several processes are unable to move forward because they are each waiting for resources that the others have.
- A deadlock-avoiding method is the Banker's Algorithm. Through the banker's algorithm, we can determine the safe sequence of the processes. We can determine whether allocating resources to a process will be "safe" or not.
- Base Logic of the algorithm:
  Need[i] = Max[i] - Allocation[i]

### Advantages of Banker's Algorithm:

- Avoids Deadlock - A deadlock is a situation in which two computer programs partaking the identical resource are effectively averting each other from accessing the resource, performing in both programs ending to perform
- Less Restrictive than deadlock prevention

### Disadvantages of Banker's Algorithm:

- It only works with a fixed number of resources and processes

- It doesn't allow processes to exchange their maximum needs while processing

## CODE:

```
import threading
import pprint

def is_safe_state(available, max_claim, allocated):
    num_processes=len(allocated)
    num_resources=len(available)

    # Initialize work and finish arrays
    work=available[:]
    finish =[False] * num_processes

    safe_sequence=[]

    #loop through processes until all are finished or no safe
sequence exists
    while True:
        #finding an index i such that process i is not finished
and needs <= available
        found= False
        for i in range(num_processes):
            if not finish[i] and all(need + work[j] >=
max_claim[i][j] for j, need in enumerate(allocated[i])):
                for j in range(num_resources):
                    work[j]+= allocated[i][j]
                finish[i]=True
                safe_sequence.append(i)
                found=True
        if not found:
            break
```

```
    #check if all processes are finished, return safe sequence
    if all(finish):
        return safe_sequence
    else:
        return None

available_resources = [1, 5, 2]
max_needed_resources = [
    [7, 5, 3],
    [3, 2, 2],
    [3, 1, 4],
    [4, 2, 2]
]
allocated_resources = [
    [0, 1, 0],
    [2, 0, 0],
    [3, 0, 2],
    [2, 1, 1]
]

# Deadlock resources
# max_needed_resources = [
#     [7, 5, 3],
#     [3, 2, 2],
#     [9 ,0 ,2],
#     [4, 2, 2]
# ]

safe_sequence = is_safe_state(available_resources,
max_needed_resources, allocated_resources)

if safe_sequence:
    print("Safe sequence:", safe_sequence)
else:
    print("No safe sequence found.")
```

## OUTPUT:

```
PS D:\OS_lab> & C:/Users/Divyanshu/AppData/Local/Microsoft/
Safe sequence: [1, 2, 3, 0]
PS D:\OS_lab> & C:/Users/Divyanshu/AppData/Local/Microsoft/
No safe sequence found.
PS D:\OS_lab>
```

## CONCLUSION:

Thus,we have successfully implemented Banker’s Algorithm for the prevention of deadlock.The Banker's Algorithm is a resource allocation and deadlock avoidance algorithm that ensures that the system is always in a safe state by dynamically checking resource requests against available resources.