NAME OF STUDENT: ______________________ID No: ___________________

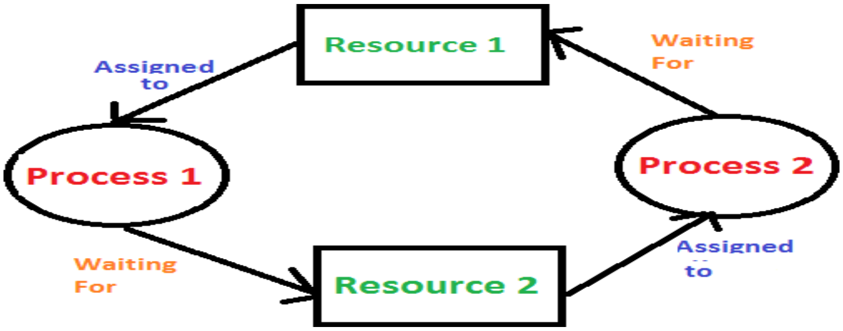
Lab no 12

Banker Algorithm for deadlock prevation

Objectives:

- What is deadlock?
- How to prevent deadlock?
- Implementation of banker algorithm for prevent deadlock.

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.



Deadlock can arise if the **following four conditions hold simultaneously (Necessary Conditions)**

Mutual Exclusion: Two or more resources are non-shareable (Only one process can use at a time)

Hold and Wait: A process is holding at least one resource and waiting for resources.

No Preemption: A resource cannot be taken from a process unless the process releases the resource.

Circular Wait: A set of processes are waiting for each other in circular form.

Banker's Algorithm

The banker‟s algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for 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.

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

1) Let Work and Finish be vectors of length „m‟ and „n‟ respectively.
Initialize: Work = Available
Finish[i] = false; for i=1, 2, 3, 4….n
2) Find an i such that both
a) Finish[i] = false

```
b) Needi <= Work
if no such i exists goto step (4)
3) Work = Work + Allocation[i]
Finish[i] = true
goto step (2)
4) if Finish [i] = true for all i
then the system is in a safe state
```

```
# Banker's Algorithm In Pyton

# Number of processes
P = 5

# Number of resources
R = 3

# Function to find the need of each
process
def calculateNeed(need, maxm, allot):

    # Calculating Need of each P
    for i in range(P):
        for j in range(R):

            # Need of instance = maxm
instance -
            # allocated instance
            need[i][j] = maxm[i][j] -
allot[i][j]

# Function to find the system is in
# safe state or not
```

NAME OF STUDENT: ______________________ID No: ___________________

```
def isSafe(processes, avail, maxm,

allot):
    need = []
    for i in range(P):
        l = []
        for j in range(R):
            l.append(0)
        need.append(l)

    # Function to calculate need matrix
    calculateNeed(need, maxm, allot)

    # Mark all processes as infinish
    finish = [0] * P

    # To store safe sequence
    safeSeq = [0] * P

    # Make a copy of available resources
    work = [0] * R
    for i in range(R):
        work[i] = avail[i]

    # While all processes are not finished
    # or system is not in safe state.
    count = 0
    while (count < P):

        # Find a process which is not finish
        # and whose needs can be satisfied
        # with current work[] resources.
        found = False
        for p in range(P):

            # First check if a process is
finished,
            # if no, go for next condition
            if (finish[p] == 0):

                # Check if for all resources
                # of current P need is less
                # than work
                for j in range(R):
                    if (need[p][j] > work[j]):
                        break

                # If all needs of p were
satisfied.
                if (j == R - 1):

                    # Add the allocated
resources of
                    # current P to the
available/work
                    # resources i.e.free the
resources
                    for k in range(R):
                        work[k] += allot[p][k]

                    # Add this process to safe
sequence.
                    safeSeq[count] = p
                    count += 1

                    # Mark this p as finished
                    finish[p] = 1

                    found = True

        # If we could not find a next
process
        # in safe sequence.
        if (found == False):
            print("System is not in safe
state")
            return False

    # If system is in safe state then
    # safe sequence will be as below
    print("System is in safe state.",
              "\nSafe sequence is: ", end = "
")
    print(*safeSeq)

    return True

# Driver code
if __name__ =="__main__":

    processes = [0, 1, 2, 3, 4]

    # Available instances of resources
    avail = [3, 3, 2]
```

```
# Maximum R that can be allocated
# to processes
maxm = [[7, 5, 3], [3, 2, 2],
        [9, 0, 2], [2, 2, 2],
        [4, 3, 3]]

# Resources allocated to processes
allot = [[0, 1, 0], [2, 0, 0],
         [3, 0, 2], [2, 1, 1],
         [0, 0, 2]]

# Check system is in safe state or not
isSafe(processes, avail, maxm, allot)
```

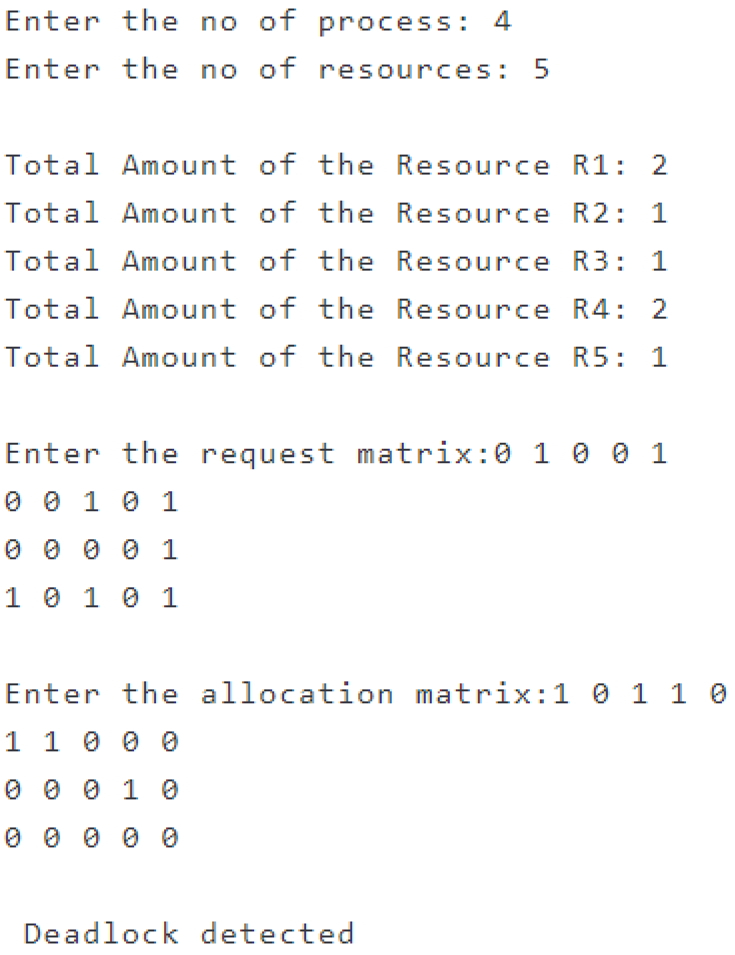
TASK:

- Design a Safety Algorithm Program for Deadlock Prevention in Python.

```
class DeadLockDetection():
    def main(self):
        processes = int(input("number of processes : "))
        resources = int(input("number of resources : "))
        max_resources = [int(i) for i in input("maximum resources : ").split()]

        print("\n-- allocated resources for each process --")
        currently_allocated = [[int(i) for i in input(f"process {j + 1} :
").split()] for j in range(processes)]

        print("\n-- maximum resources for each process --")
        max_need = [[int(i) for i in input(f"process {j + 1} : ").split()] for j in
range(processes)]

        allocated = [0] * resources
        for i in range(processes):
            for j in range(resources):
                allocated[j] += currently_allocated[i][j]
        print(f"\ntotal allocated resources : {allocated}")

        available = [max_resources[i] - allocated[i] for i in range(resources)]

        running = [True] * processes
        count = processes
        while count != 0:
            safe = False
            for i in range(processes):
                if running[i]:
                    executing = True
                    for j in range(resources):
                        if max_need[i][j] - currently_allocated[i][j] > available[j]:
                            executing = False
                            break
                    if executing:
                        print(f" Total amount of the Resource R{i}:{count}")
                        running[i] = False
                        count -= 1
                        safe = True
                        for j in range(resources):
                            available[j] += currently_allocated[i][j]
                        break
            if not safe:
                print("deadlock detected")
                break

            print("No deadlock detected")


if __name__ == '__main__':
    d = DeadLockDetection()
    d.main()
```

NAME OF STUDENT: ______________________ID No: ___________________

Output:



```
Enter the no of process: 4
Enter the no of resources: 5

Total Amount of the Resource R1: 2
Total Amount of the Resource R2: 1
Total Amount of the Resource R3: 1
Total Amount of the Resource R4: 2
Total Amount of the Resource R5: 1

Enter the request matrix:0 1 0 0 1
0 0 1 0 1
0 0 0 0 1
1 0 1 0 1

Enter the allocation matrix:1 0 1 1 0
1 1 0 0 0
0 0 0 1 0
0 0 0 0 0

 Deadlock detected
```