

EXPERIMENT NO. 5

AIM: Write a program to demonstrate Deadlock Avoidance through Banker's Algorithm

RESOURCES REQUIRED:

H/W Requirements: P-IV and above, Ram 128 MB, Printer, Internet Connection.

S/W Requirements: Python Compiler.

THEORY:

When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system.

When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. If it will, the resources are allocated; otherwise, the process must wait until some other process releases enough resources.

Several data structures must be maintained to implement the banker's algorithm. These data structures encode the state of the resource-allocation system.

We need the following data structures, where n is the number of processes in the system and m is the number of resource types:

- ❖ Available – A vector of length m indicates the number of available resources of each type. If Available[j] equals k , then k instances of resource type R_j are available.
- ❖ Max. An $n \times m$ matrix defines the maximum demand of each process. If Max[i][j] equals k , then process P_i may request at most k instances of resource type R_j .
- ❖ Allocation. An $n \times m$ matrix defines the number of resources of each type currently allocated to each process. If Allocation[i][j] equals k , then process P_i is currently allocated k instances of resource type R_j .
- ❖ Need. An $n \times m$ matrix indicates the remaining resource need of each process. If Need[i][j] equals k , then process P_i may need k more instances of resource type R_j to complete its task. Note that Need[i][j] equals Max[i][j] – Allocation[i][j].

This algorithm can be described as follows:

(1) Let *Work* and *Finish* be vectors of length *m* and *n*, respectively. Initialize *Work* = *Available* and *Finish*[*i*] = false for *i* = 0, 1, ..., *n* - 1.

(2) Find an index *i* such that both

- *Finish*[*i*] == false
- *Need*_{*i*} ≤ *Work*
- If no such *i* exists, go to step 4.

(3) *Work* = *Work* + *Allocation* _{*i*}

Finish[*i*] = true

Go to step 2.

(4) If *Finish*[*i*] == true for all *i*, then the system is in a safe state.

CONCLUSION: Hence, we have demonstrated a program on deadlock avoidance through Banker's Algorithm.

CODE:

def banker():

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):

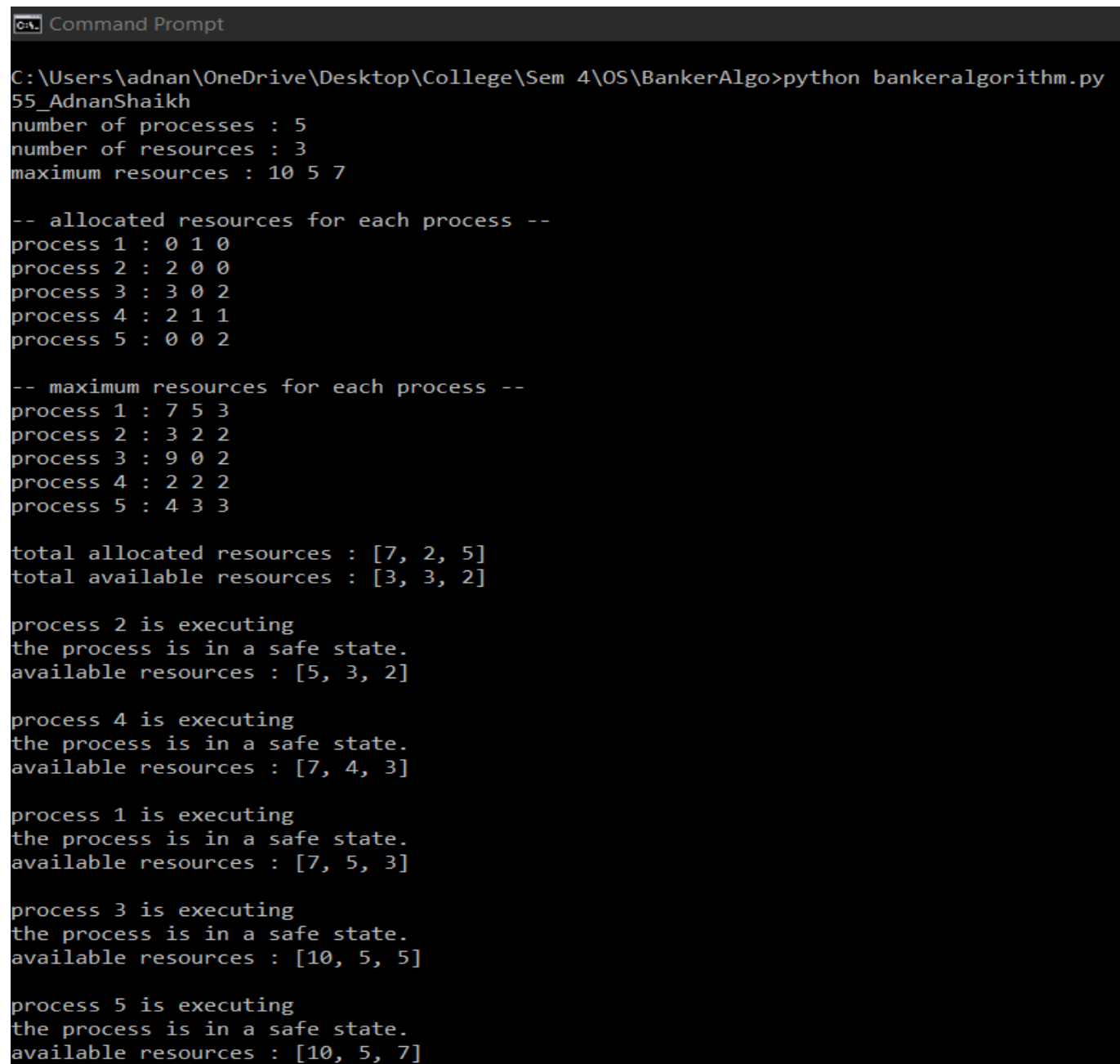
```

    allocated[j] += currently_allocated[i][j]
print(f"\ntotal allocated resources : {allocated}")
available = [max_resources[i] - allocated[i] for i in range(resources)]
print(f"total available resources : {available}\n")
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"process {i + 1} is executing")
                running[i] = False
                count -= 1
                safe = True
                for j in range(resources):
                    available[j] += currently_allocated[i][j]
                break
    if not safe:
        print("the processes are in an unsafe state.")
        break
print(f"the process is in a safe state.\navailable resources : {available}\n")

```

```
if __name__ == '__main__':  
    print("55_AdnanShaikh")  
    banker()
```

OUTPUT:



```
Command Prompt  
C:\Users\adnan\OneDrive\Desktop\College\Sem 4\OS\BankerAlgo>python bankeralgorithm.py  
55_AdnanShaikh  
number of processes : 5  
number of resources : 3  
maximum resources : 10 5 7  
  
-- allocated resources for each process --  
process 1 : 0 1 0  
process 2 : 2 0 0  
process 3 : 3 0 2  
process 4 : 2 1 1  
process 5 : 0 0 2  
  
-- maximum resources for each process --  
process 1 : 7 5 3  
process 2 : 3 2 2  
process 3 : 9 0 2  
process 4 : 2 2 2  
process 5 : 4 3 3  
  
total allocated resources : [7, 2, 5]  
total available resources : [3, 3, 2]  
  
process 2 is executing  
the process is in a safe state.  
available resources : [5, 3, 2]  
  
process 4 is executing  
the process is in a safe state.  
available resources : [7, 4, 3]  
  
process 1 is executing  
the process is in a safe state.  
available resources : [7, 5, 3]  
  
process 3 is executing  
the process is in a safe state.  
available resources : [10, 5, 5]  
  
process 5 is executing  
the process is in a safe state.  
available resources : [10, 5, 7]
```