Date - 9/9/2020

Experiment No. 4

Banker's Algorithm

Aim: To write a program to implement banker's algorithm for deadlock avoidanc

Theory: Banker's Algorithm is a deadlock avoidance algorithm. It is also used for deadlock detection. This algorithm tells that if any system can go into a deadlock or not by analyzing the currently allocated resources and the resources required by it in the future.

Data Structures required are::

1. Available:

It is a 1-D array that tells the number of each resource type (instance of resource type) currently available.

2. Max

It is a 2-D array that tells the maximum number of each resource type required by a process for successful execution

3. Allocation

It is a 2-D array that tells the number of types of each resource type that has been allocated to the process.

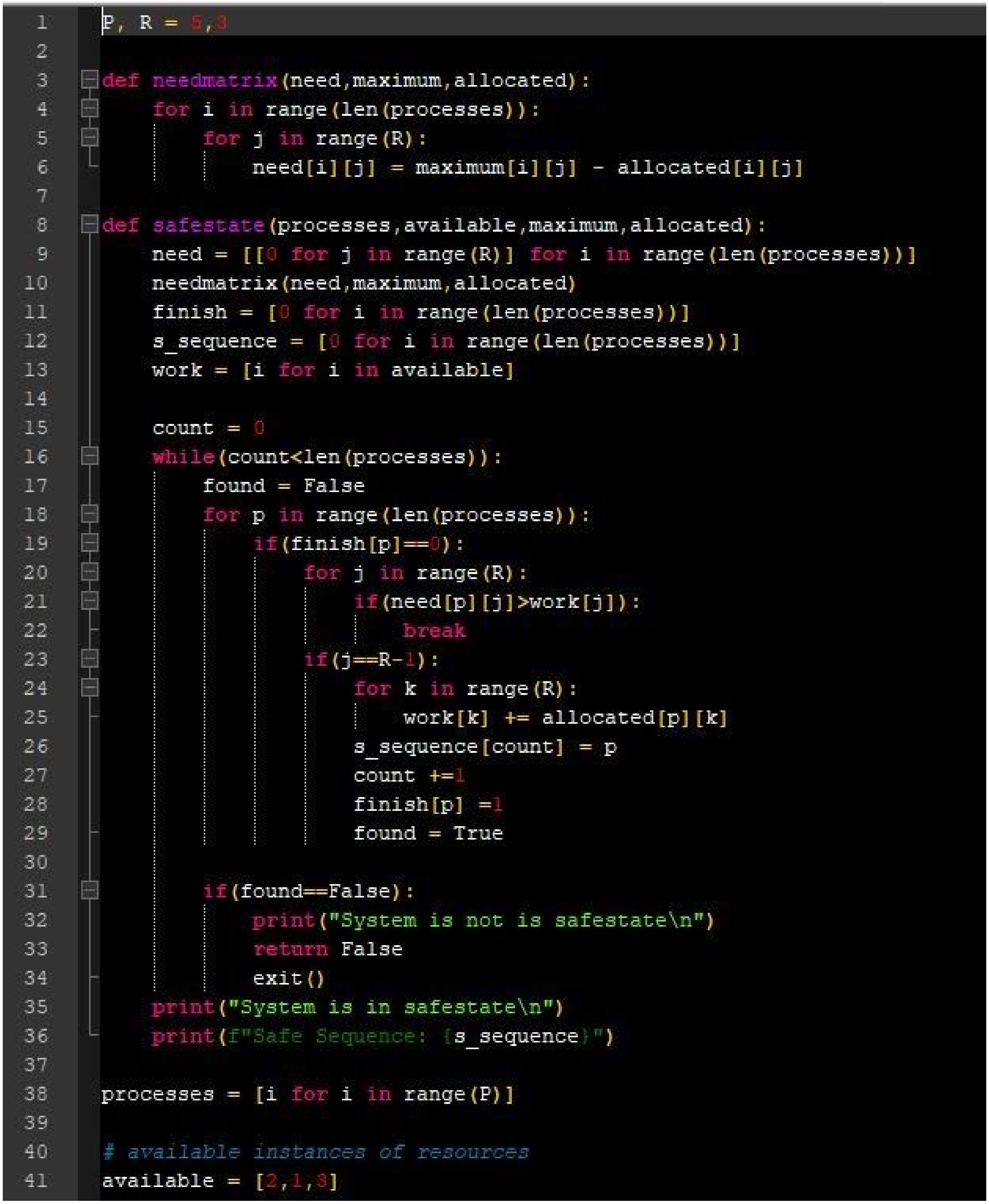
4. Need

It is a 2-D array that tells the number of remaining instances of each resource type required for execution.

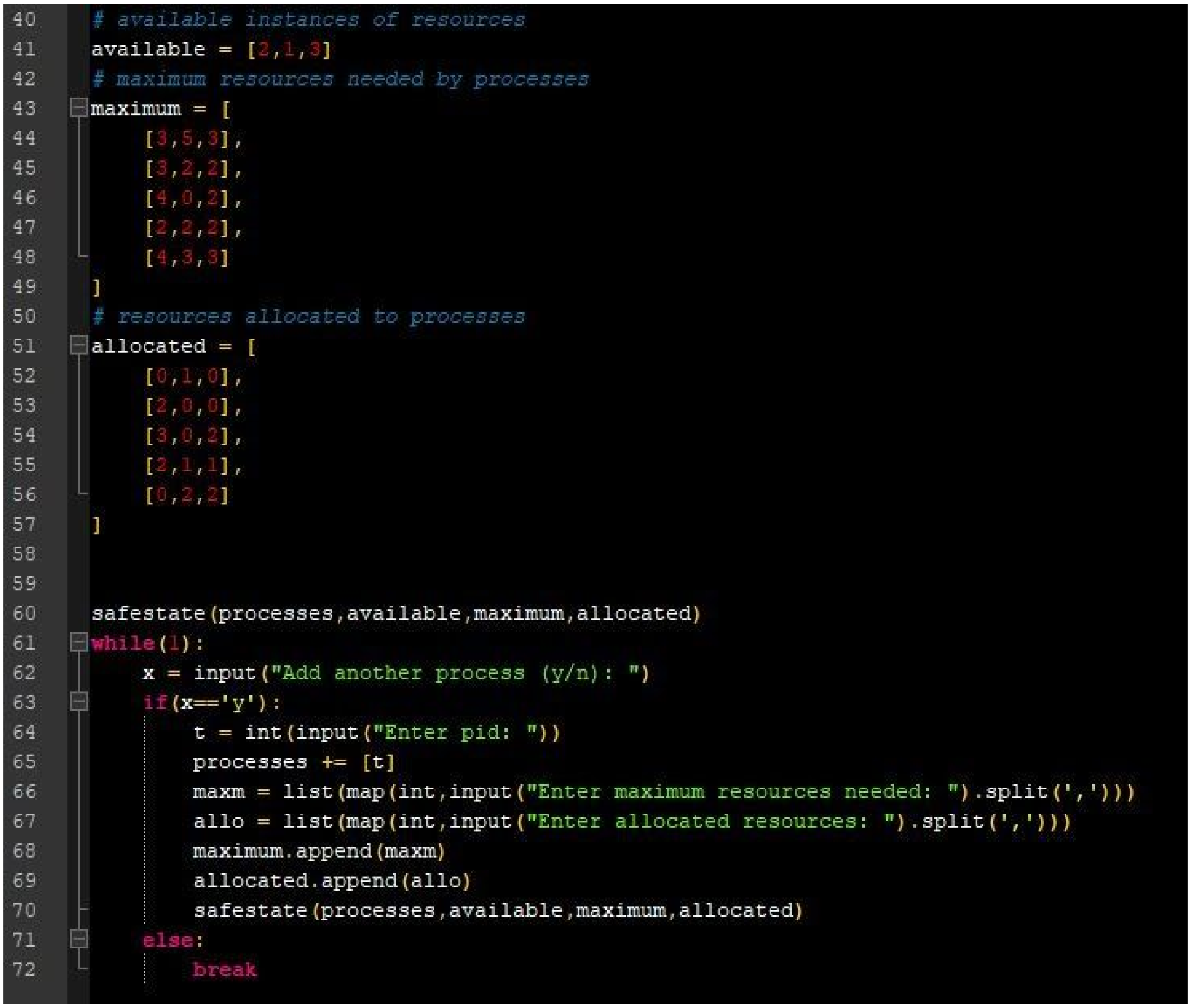
Algorithm:

```
Step 1: Start.
Step 2: Set number of processes, P and resources, R.
Step 3: Initialise 1D arrays processes and available processes = array(pid of P number of processes ) available = array(available instances of R number of resources)
Step 4: Initialise 2D arrays of maximum and allocated resources maximum[i][j] = number of instances of R resources required by pid i allocated[i][j] = number of allocated instances of R resources required by pid i
Step 5: Calculate need matrix <=>  need[i][j] = maximum[i][j] - allocated[i][j]
Step 6: Set finish[i] = 0 for all i till P
Step 7: Find i such that finish[i] = False
            If need[i][j] < allocated[i][j]
            Set finish[i] = 1
            Add pid(i) to array(safeseq)
            Repeat
Step 8: if(finish):
                print(safeseq)
            else:
                print("System not in safe state")
Step 9: Stop the program.
```

Program:

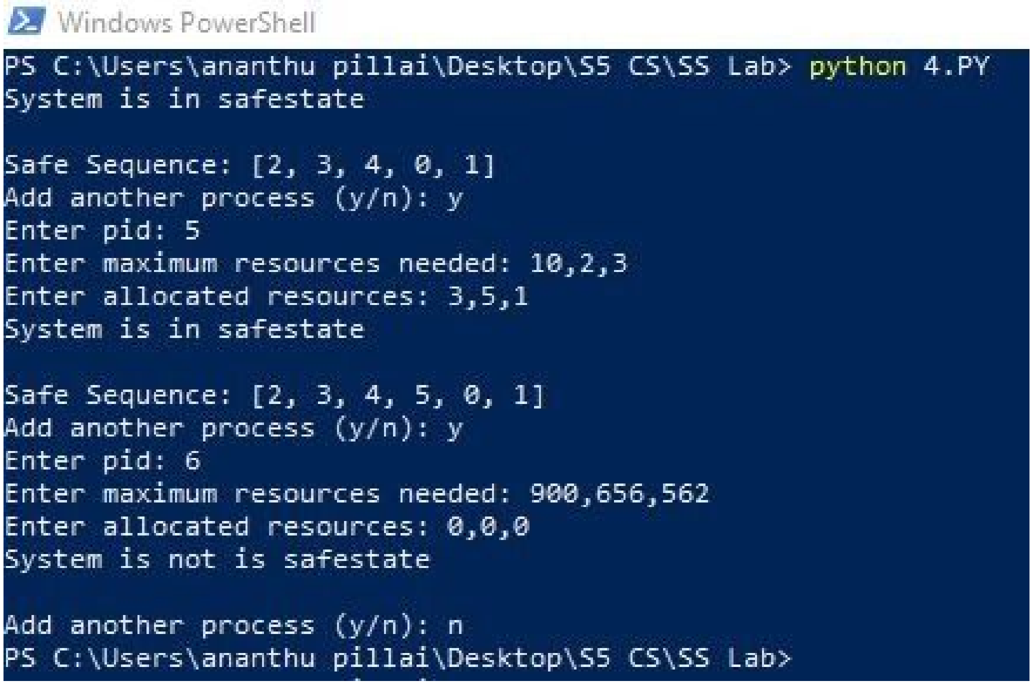


```
P, R = 5,3

def needmatrix(need,maximum,allocated):
    for i in range(len(processes)):
        for j in range(R):
            need[i][j] = maximum[i][j] - allocated[i][j]

def safestate(processes,available,maximum,allocated):
    need = [[0 for j in range(R)] for i in range(len(processes))]
    needmatrix(need,maximum,allocated)
    finish = [0 for i in range(len(processes))]
    s_sequence = [0 for i in range(len(processes))]
    work = [i for i in available]

    count = 0
    while(count<len(processes)):
        found = False
        for p in range(len(processes)):
            if(finish[p]==0):
                for j in range(R):
                    if(need[p][j]>work[j]):
                        break
                if(j==R-1):
                    for k in range(R):
                        work[k] += allocated[p][k]
                    s_sequence[count] = p
                    count +=1
                    finish[p] =1
                    found = True

        if(found==False):
            print("System is not is safestate\n")
            return False
            exit()
    print("System is in safestate\n")
    print(f"Safe Sequence: {s_sequence}")

processes = [i for i in range(P)]

# available instances of resources
available = [2,1,3]
```



```python
# available instances of resources
available = [2,1,3]
# maximum resources needed by processes
maximum = [
    [3,5,3],
    [3,2,2],
    [4,0,2],
    [2,2,2],
    [4,3,3]
]
# resources allocated to processes
allocated = [
    [0,1,0],
    [2,0,0],
    [3,0,2],
    [2,1,1],
    [0,2,2]
]


safestate(processes,available,maximum,allocated)
while(1):
    x = input("Add another process (y/n): ")
    if(x=='y'):
        t = int(input("Enter pid: "))
        processes += [t]
        maxm = list(map(int,input("Enter maximum resources needed: ").split(',')))
        allo = list(map(int,input("Enter allocated resources: ").split(',')))
        maximum.append(maxm)
        allocated.append(allo)
        safestate(processes,available,maximum,allocated)
    else:
        break
```

Output:



```
PS C:\Users\ananthu pillai\Desktop\S5 CS\SS Lab> python 4.PY
System is in safestate

Safe Sequence: [2, 3, 4, 0, 1]
Add another process (y/n): y
Enter pid: 5
Enter maximum resources needed: 10,2,3
Enter allocated resources: 3,5,1
System is in safestate

Safe Sequence: [2, 3, 4, 5, 0, 1]
Add another process (y/n): y
Enter pid: 6
Enter maximum resources needed: 900,656,562
Enter allocated resources: 0,0,0
System is not is safestate

Add another process (y/n): n
PS C:\Users\ananthu pillai\Desktop\S5 CS\SS Lab>
```

Result:

The program to simulate the banker's algorithm for deadlock avoidance has been implemented and simulated successfully.