K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: A2	**Roll No.: 1911027**
Experiment No. 07	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

TITLE: Simulate Bankers Algorithm for Deadlock Avoidance

AIM: Implementation of Banker's Algorithm for Deadlock Avoidance

Expected Outcome of Experiment:

CO 3. To understand the concepts of process synchronization and deadlock.

Books/ Journals/ Websites referred:

1. **Silberschatz A., Galvin P., Gagne G. "Operating Systems Principles", Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate "Operating Systems" McGraw Hill Third Edition.**
3. **William Stallings, "Operating System Internal & Design Principles", Pearson.**
4. **Andrew S. Tanenbaum, "Modern Operating System", Prentice Hall.**

Pre Lab/ Prior Concepts:

Knowledge of deadlocks and all deadlock avoidance methods.

Description of the application to be implemented:

The Banker's algorithm is a resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra. 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. When a new process is created in a computer system, the process must provide all types of information to the operating system like upcoming processes, requests for their resources, counting them, and delays. Based on these criteria, the operating system decides which process sequence should be executed or waited so that no deadlock occurs in a system. Therefore, it is also known as deadlock avoidance algorithm or deadlock detection in the operating system.

Advantages:

- It contains various resources that meet the requirements of each process.
- Each process should provide information to the operating system for upcoming resource requests, the number of resources, and how long the resources will be held.
- It helps the operating system manage and control process requests for each type of resource in the computer system.
- The algorithm has a Max resource attribute that represents indicates each process can hold the maximum number of resources in a system.

Disadvantages:

- It requires a fixed number of processes, and no additional processes can be started in the system while executing the process.
- The algorithm does no longer allows the processes to exchange its maximum needs while processing its tasks.
- Each process has to know and state their maximum resource requirement in advance for the system.
- The number of resource requests can be granted in a finite time, but the time limit for allocating the resources is one year.

DATA STRUCTURES: Following Data structures are used to implement the Banker's Algorithm: Let 'n' be the number of processes in the system and 'm' be the number of resources types.

Available: It is a 1-d array of size 'm' indicating the number of available resources of each type. Available[j] = k means there are 'k' instances of resource type Rj.

Max: It is a 2-d array of size 'n*m' that defines the maximum demand of each process in a system. Max[i, j] = k means process Pi may request at most 'k' instances of resource type Rj.

Allocation: It is a 2-d array of size 'n*m' that defines the number of resources of each type currently allocated to each process. Allocation[i, j] = k means process Pi is currently allocated 'k' instances of resource type Rj.

Need: It is a 2-d array of size 'n*m' that indicates the remaining resource need of each process. Need [i, j] = k means process Pi currently need 'k' instances of resource type Rj. Need [i, j] = Max [i, j] – Allocation [i, j].

Allocationi specifies the resources currently allocated to process Pi and Needi specifies the additional resources that process Pi may still request to complete its task. Banker's algorithm consists of Safety algorithm and Resource request algorithm.

Safety Algorithm: 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) $Need_i$ <= 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
```

Implementation details: (printout of code)

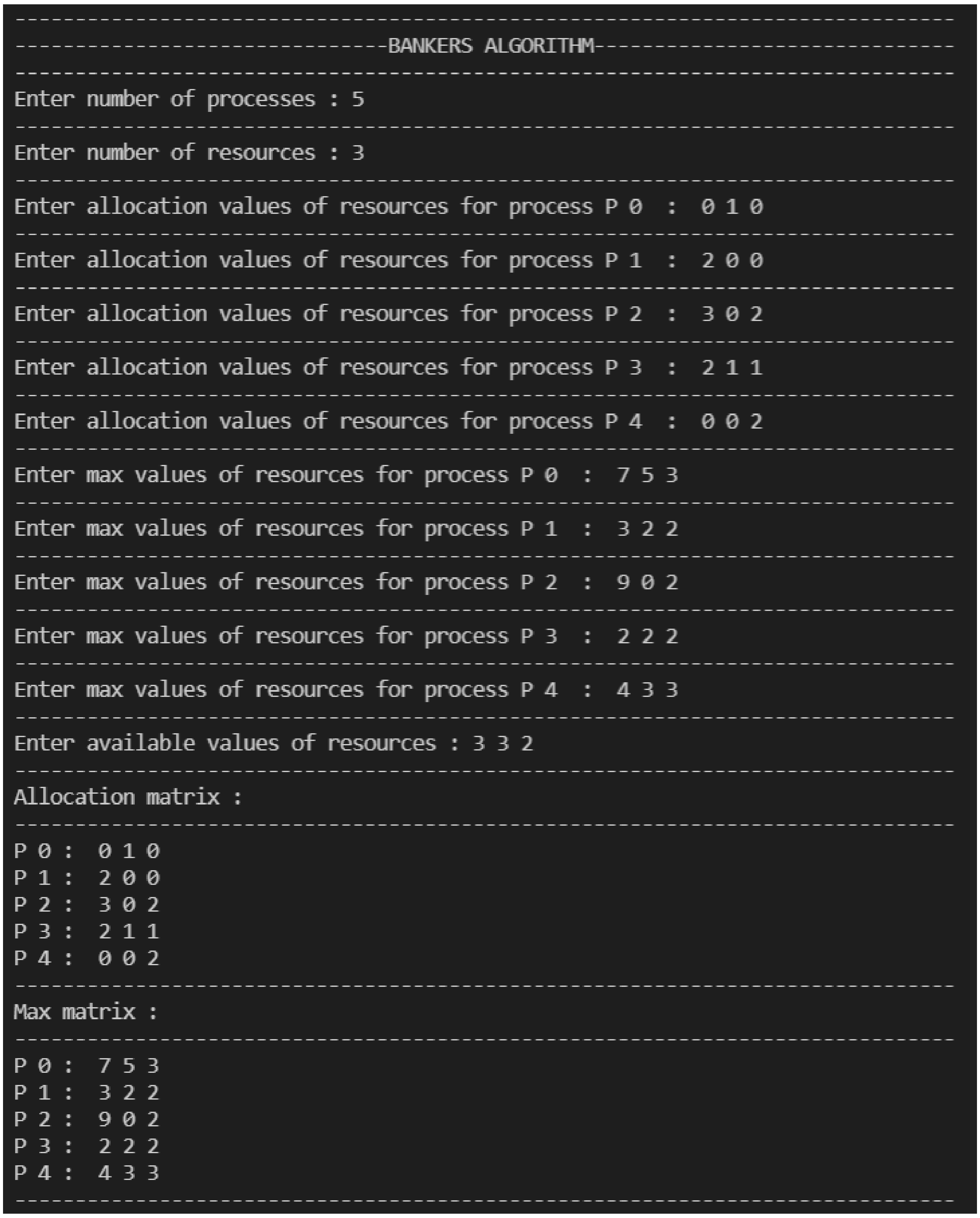
Code:

```
import copy
def checker(need,work):
    for i,j in zip(need,work):
        if(i>j):
            return False
    return True
def exitter(fin):
    for i in fin:
        if(i==False):
            return False
    return True
print("------------------------------------------------------------------
------------")
print("-------------------------------BANKERS ALGORITHM------------------
------------")
print("------------------------------------------------------------------
------------")
m=int(input("Enter number of processes : "))
print("------------------------------------------------------------------
------------")
n=int(input("Enter number of resources : "))
print("------------------------------------------------------------------
------------")
allocation_matrix=[]
for i in range(0,m):
    print("Enter allocation values of resources for process P",i," :
",end=" ")
    inp=list(input().split())
    print("--------------------------------------------------------------
----------------")
    temp=[]
    for i in inp:
        temp.append(int(i))
    allocation_matrix.append(temp)
max_matrix=[]
for i in range(0,m):
    print("Enter max values of resources for process P",i," : ",end=" ")
    inp=list(input().split())
    print("--------------------------------------------------------------
----------------")
    temp=[]
    for i in inp:
        temp.append(int(i))
    max_matrix.append(temp)
available_matrix=[]
inp=list(input("Enter available values of resources : ").split())
print("------------------------------------------------------------------
------------")
for i in inp:
    available_matrix.append(int(i))
print("Allocation matrix : ")
print("------------------------------------------------------------------
------------")
for i in range(0,m):
    print("P",i,": ",end=" ")
    for j in range(0,n):
        print(allocation_matrix[i][j],end=" ")
    print()
print("------------------------------------------------------------------
------------")
print("Max matrix : ")
print("------------------------------------------------------------------
------------")
for i in range(0,m):
    print("P",i,": ",end=" ")
    for j in range(0,n):
        print(max_matrix[i][j],end=" ")
    print()
print("------------------------------------------------------------------
------------")
print("Available matrix : ")
print("------------------------------------------------------------------
------------")
for i in range(0,n):
    print(available_matrix[i],end=" ")
print("\n----------------------------------------------------------------
--------------")
```

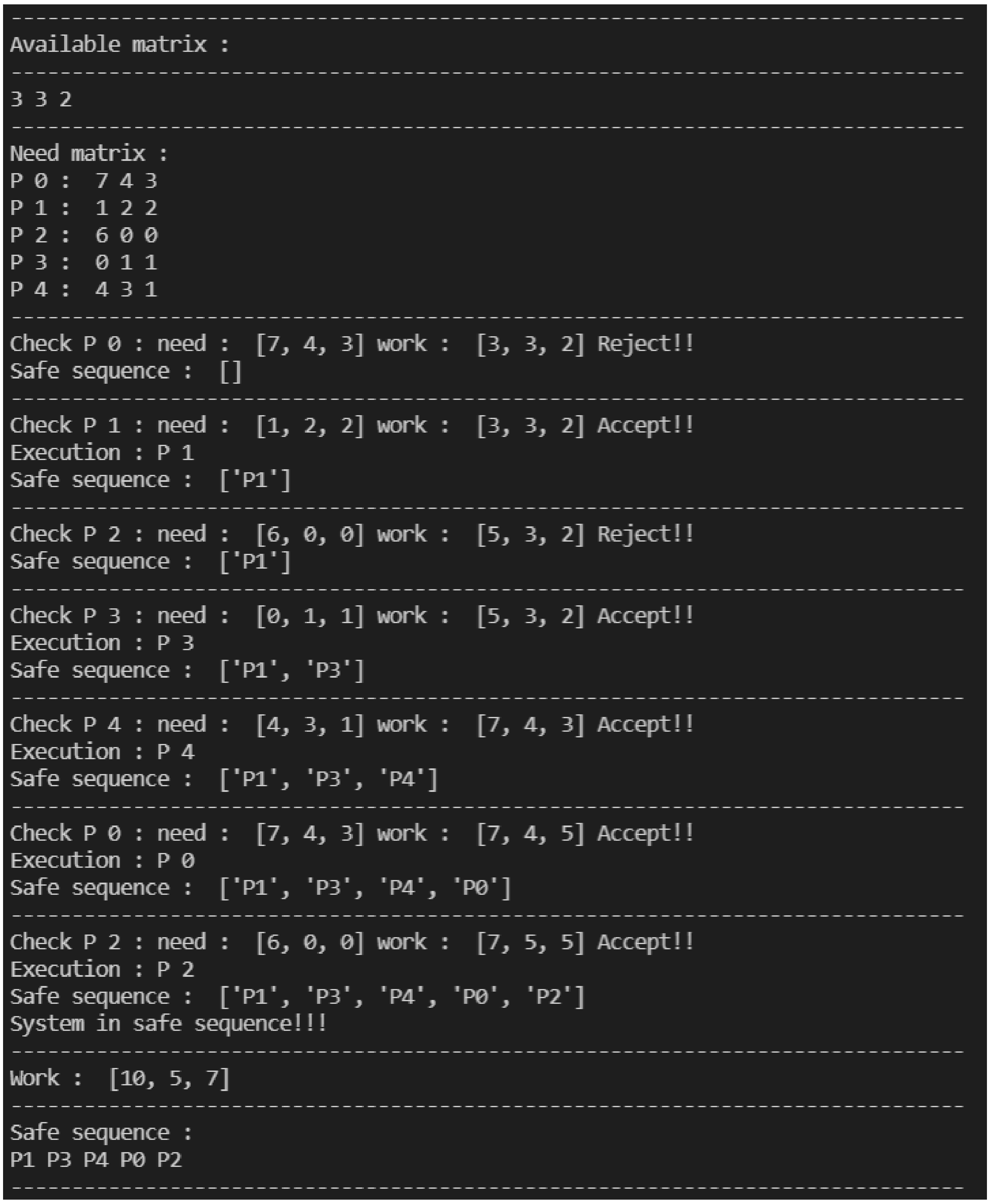
```python
need_matrix=[]
for i,j in zip(max_matrix,allocation_matrix):
    temp=[]
    for k,l in zip(i,j):
        temp.append(k-l)
    need_matrix.append(temp)
print("Need matrix : ")
for i in range(0,m):
    print("P",i,": ",end=" ")
    for j in range(0,n):
        print(need_matrix[i][j],end=" ")
    print()
print("------------------------------------------------------------------
------------")
work=copy.deepcopy(available_matrix)
finish=[]
for i in range(0,m):
    finish.append(False)
flag=True
i=0
count=0
safe_sequence=[]
temp=copy.deepcopy(m)
while(flag):
    if(finish[i]==False):
        print("Check P",i,": need : ",need_matrix[i],"work : ",work,end="
")
    if(checker(need_matrix[i],work) and finish[i]==False):
        print("Accept!!")
        string="P"+str(i)
        safe_sequence.append(string)
        print("Execution : P",i)
        print("Safe sequence : ",safe_sequence)
        for j in range(0,n):
            work[j]=work[j]+allocation_matrix[i][j]
        finish[i]=True
        if(exitter(finish)):
            print("System in safe sequence!!!")
            print("------------------------------------------------------
------------------------")
            break
        else:
            i=(i+1)%m
        temp=temp-1
        count=0
        print("-----------------------------------------------------------
--------------------")
    else:
        if(finish[i]==False):
            print("Reject!!")
            count=count+1
            print("Safe sequence : ",safe_sequence)
            print("-------------------------------------------------------
------------------------")
        i=(i+1)%m
    if(count==temp):
        flag=False
        print("No safe state obtained!!!")
        print("-----------------------------------------------------------
--------------------")
if(flag):
    print("Work : ",work)
    print("---------------------------------------------------------------
----------------")
    print("Safe sequence : ")
    for i in safe_sequence:
        print(i,end=" ")
    print("\n-------------------------------------------------------------
------------------")
```

Output:

If safe state can be obtained:

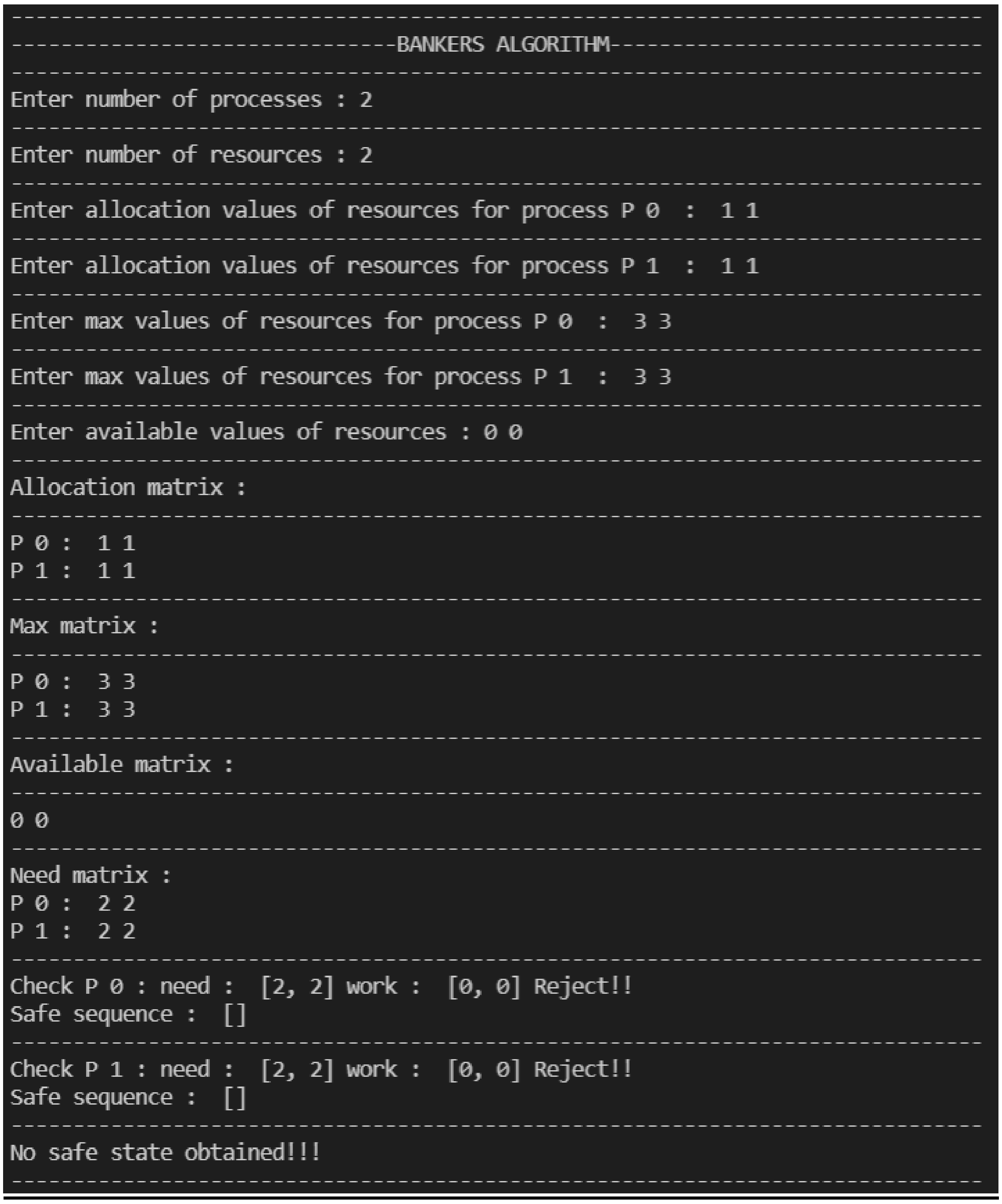


```
------------------------------------------------------------------------------
-----------------------------BANKERS ALGORITHM-----------------------------
------------------------------------------------------------------------------
Enter number of processes : 5
------------------------------------------------------------------------------
Enter number of resources : 3
------------------------------------------------------------------------------
Enter allocation values of resources for process P 0  :  0 1 0
------------------------------------------------------------------------------
Enter allocation values of resources for process P 1  :  2 0 0
------------------------------------------------------------------------------
Enter allocation values of resources for process P 2  :  3 0 2
------------------------------------------------------------------------------
Enter allocation values of resources for process P 3  :  2 1 1
------------------------------------------------------------------------------
Enter allocation values of resources for process P 4  :  0 0 2
------------------------------------------------------------------------------
Enter max values of resources for process P 0  :  7 5 3
------------------------------------------------------------------------------
Enter max values of resources for process P 1  :  3 2 2
------------------------------------------------------------------------------
Enter max values of resources for process P 2  :  9 0 2
------------------------------------------------------------------------------
Enter max values of resources for process P 3  :  2 2 2
------------------------------------------------------------------------------
Enter max values of resources for process P 4  :  4 3 3
------------------------------------------------------------------------------
Enter available values of resources : 3 3 2
------------------------------------------------------------------------------
Allocation matrix :
------------------------------------------------------------------------------
P 0 :  0 1 0
P 1 :  2 0 0
P 2 :  3 0 2
P 3 :  2 1 1
P 4 :  0 0 2
------------------------------------------------------------------------------
Max matrix :
------------------------------------------------------------------------------
P 0 :  7 5 3
P 1 :  3 2 2
P 2 :  9 0 2
P 3 :  2 2 2
P 4 :  4 3 3
------------------------------------------------------------------------------
```



```
---------------------------------------------------------------------------
Available matrix :
---------------------------------------------------------------------------
3 3 2
---------------------------------------------------------------------------
Need matrix :
P 0 :  7 4 3
P 1 :  1 2 2
P 2 :  6 0 0
P 3 :  0 1 1
P 4 :  4 3 1
---------------------------------------------------------------------------
Check P 0 : need :  [7, 4, 3] work :  [3, 3, 2] Reject!!
Safe sequence :  []
---------------------------------------------------------------------------
Check P 1 : need :  [1, 2, 2] work :  [3, 3, 2] Accept!!
Execution : P 1
Safe sequence :  ['P1']
---------------------------------------------------------------------------
Check P 2 : need :  [6, 0, 0] work :  [5, 3, 2] Reject!!
Safe sequence :  ['P1']
---------------------------------------------------------------------------
Check P 3 : need :  [0, 1, 1] work :  [5, 3, 2] Accept!!
Execution : P 3
Safe sequence :  ['P1', 'P3']
---------------------------------------------------------------------------
Check P 4 : need :  [4, 3, 1] work :  [7, 4, 3] Accept!!
Execution : P 4
Safe sequence :  ['P1', 'P3', 'P4']
---------------------------------------------------------------------------
Check P 0 : need :  [7, 4, 3] work :  [7, 4, 5] Accept!!
Execution : P 0
Safe sequence :  ['P1', 'P3', 'P4', 'P0']
---------------------------------------------------------------------------
Check P 2 : need :  [6, 0, 0] work :  [7, 5, 5] Accept!!
Execution : P 2
Safe sequence :  ['P1', 'P3', 'P4', 'P0', 'P2']
System in safe sequence!!!
---------------------------------------------------------------------------
Work :  [10, 5, 7]
---------------------------------------------------------------------------
Safe sequence :
P1 P3 P4 P0 P2
---------------------------------------------------------------------------
```

If safe state can not be obtained:



```
----------------------------------------------------------------------------
-------------------------------BANKERS ALGORITHM-----------------------------
----------------------------------------------------------------------------
Enter number of processes : 2
----------------------------------------------------------------------------
Enter number of resources : 2
----------------------------------------------------------------------------
Enter allocation values of resources for process P 0  :  1 1
----------------------------------------------------------------------------
Enter allocation values of resources for process P 1  :  1 1
----------------------------------------------------------------------------
Enter max values of resources for process P 0  :  3 3
----------------------------------------------------------------------------
Enter max values of resources for process P 1  :  3 3
----------------------------------------------------------------------------
Enter available values of resources : 0 0
----------------------------------------------------------------------------
Allocation matrix :
----------------------------------------------------------------------------
P 0 :  1 1
P 1 :  1 1
----------------------------------------------------------------------------
Max matrix :
----------------------------------------------------------------------------
P 0 :  3 3
P 1 :  3 3
----------------------------------------------------------------------------
Available matrix :
----------------------------------------------------------------------------
0 0
----------------------------------------------------------------------------
Need matrix :
P 0 :  2 2
P 1 :  2 2
----------------------------------------------------------------------------
Check P 0 : need :  [2, 2] work :  [0, 0] Reject!!
Safe sequence :  []
----------------------------------------------------------------------------
Check P 1 : need :  [2, 2] work :  [0, 0] Reject!!
Safe sequence :  []
----------------------------------------------------------------------------
No safe state obtained!!!
----------------------------------------------------------------------------
```

Conclusion: **Understood the concept of deadlocks in operating system. Also learned the concept of safe state and banker's algorithm. Implemented the banker's algorithm using python.**

Post Lab Objective Questions

1) The wait-for graph is a deadlock detection algorithm that is applicable when:
   a) All resources have a single instance
   b) All resources have multiple instances
   c) Both a and b
   d) None of the above

Ans: a) All resources have a single instance

2) Resources are allocated to the process on non-sharable basis is _
   a) Hold and Wait
   b) Mutual Exclusion
   c) No pre-emption
   d) Circular Wait

Ans: b)Mutual Exclusion

3) Which of the following approaches require knowledge of the system state?
   a) Deadlock Detection
   b) Deadlock Prevention
   c) Deadlock Avoidance
   d) All of the above

Ans: d) All of the above

4) Consider a system having 'm' resources of the same type. These resources are shared by 3 processes A, B, C which have peak time demands of 3, 4, 6 respectively. The minimum value of 'm' that

ensures that deadlock will never occur is

a) 11

b) 12

c) 13

d) 14

Ans: a) 11

Post Lab Descriptive Questions

1. Consider a system with total of 150 units of memory allocated to three processes as shown:

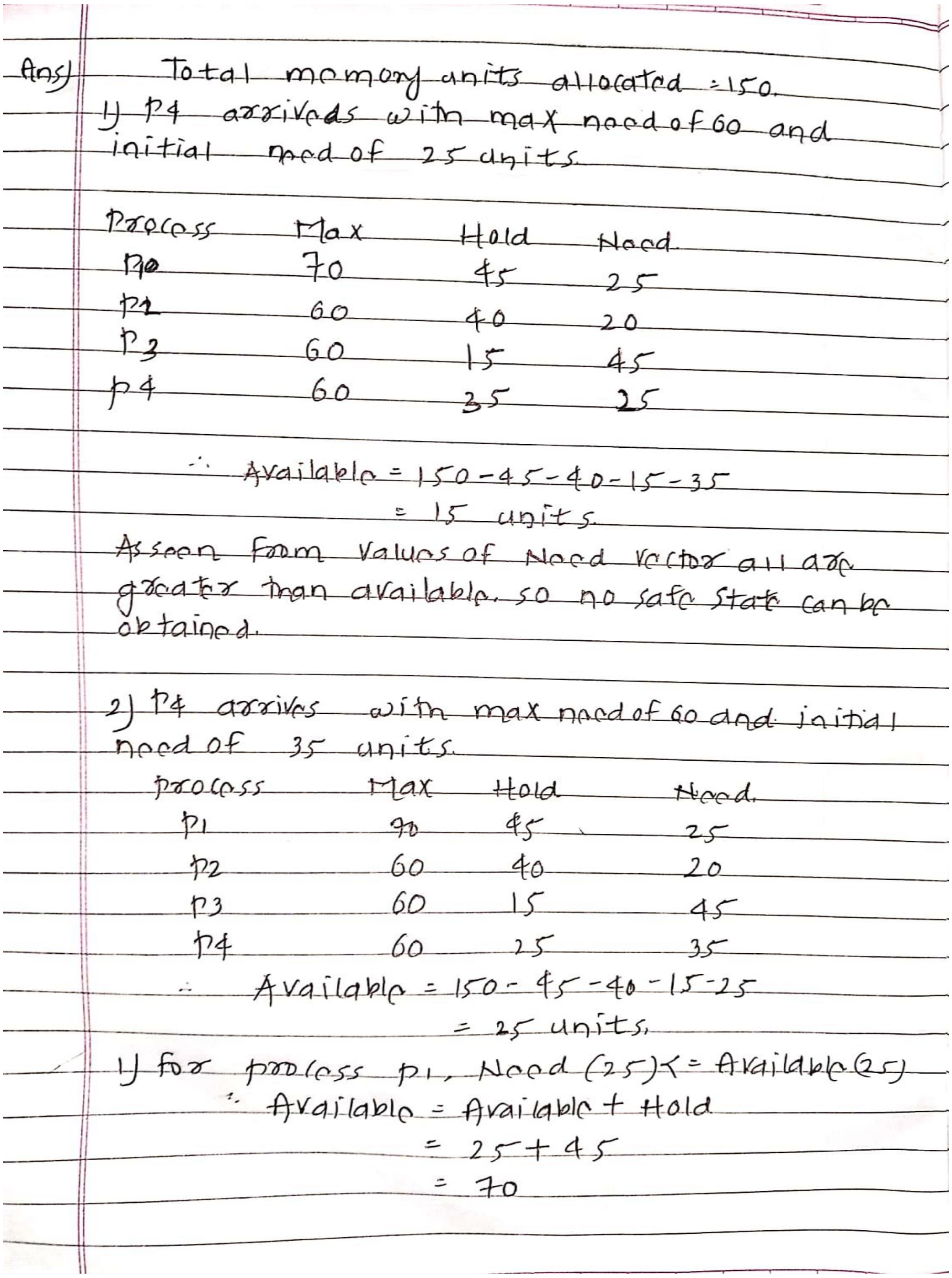
Process	**Max**	**Hold**
P1	70	45
P2	60	40
P3	60	15

Apply Banker's algorithm to determine whether it would be safe to grant each of the following request. If yes, indicate sequence of termination that could be possible.

1) The P4 process arrives with max need of 60 and initial need of 25 units.

2) The P4 process arrives with max need of 60 and initial need of 35 units.

ANS)



Ans) Total memory units allocated = 150.

1) P4 arriveds with max need of 60 and initial need of 25 units.

Process	Max	Hold	Need
P0	70	45	25
P1	60	40	20
P3	60	15	45
P4	60	35	25

∴ Available = 150 - 45 - 40 - 15 - 35
= 15 units.

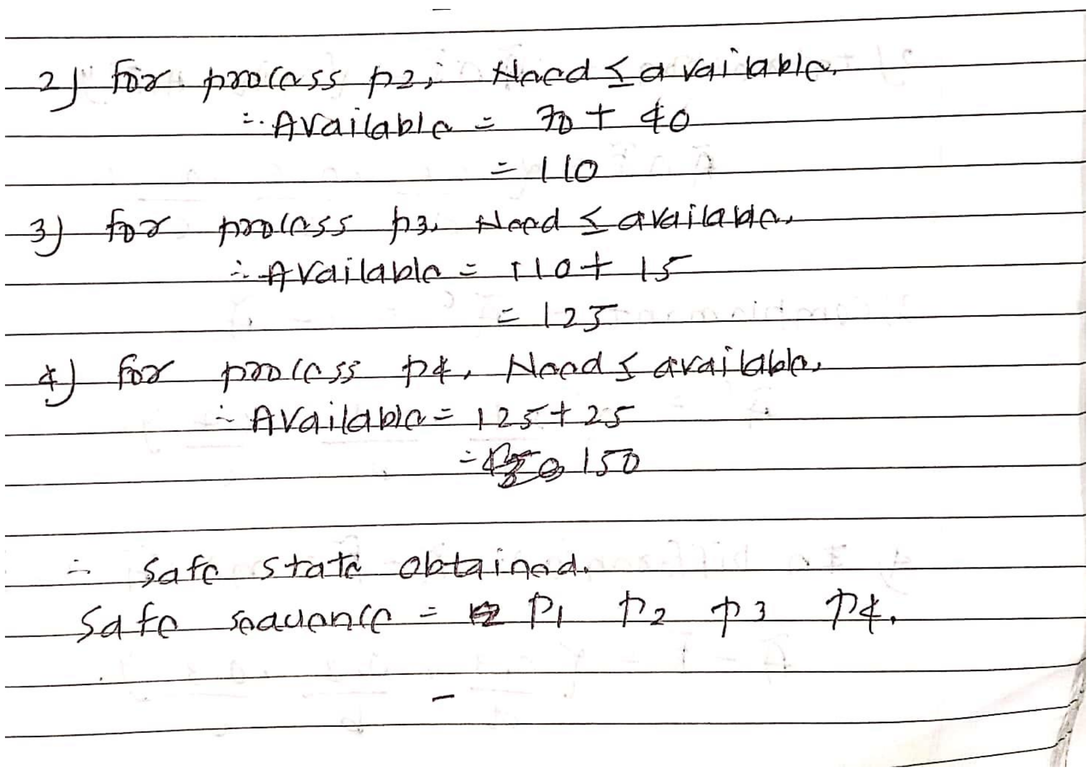
As seen from values of Need vector all are greater than available. so no safe state can be obtained.

2) P4 arrives with max need of 60 and initial need of 35 units.

process	Max	Hold	Need
P1	70	45	25
P2	60	40	20
P3	60	15	45
P4	60	25	35

∴ Available = 150 - 45 - 40 - 15 - 25
= 25 units.

1) for process P1, Need (25) <= Available (25)
∴ Available = Available + Hold
= 25 + 45
= 70



2) for process p2, Need ≤ available.
∴ Available = 70 + 40
= 110

3) for process p3, Need ≤ available.
∴ Available = 110 + 15
= 125

4) for process p4, Need ≤ available.
∴ Available = 125 + 25
= 150

∴ safe state obtained.
Safe sequence = P1 P2 P3 P4.

Date: 24 / 11 / 2021 **Signature of faculty in-charge**