

**Operating System Project 1**

**Submitted to:**

Dr. Gamal Abdel-Shafy

**Submitted by:**

**Name:** **ID:**

Ayman Hesham Mohamed16P3037

Gina Emil Attia 16P3022

Rowan Hazem Wagieh16P3023

**Brief Description:**

The program implements “Banker’salgorithm” which is one of the algorithms to prevent deadlock. It takes inputs as number of processes and number of resources and how many instance for each resource. The data structure used for this implementation uses: Available: Vector of length of number of resources. If available [j] = k, there are k instances of resource type Rj available, Maximum: number of process X number of resources matrix. If Max [i,j] = k, then process Pi may request at most k instances of resource type Rj, Allocation: number of process X number of resources. If Allocation [i,j] = k then Pi is currently allocated k instances of Rj, Need: number of process X number of resources matrix. If Need [i,j] = k, then Pi may need k more instances of Rj to complete its task, Need [i,j] = Max[i,j] – Allocation [i,j]. After taking the inputs from the user, the program makes the allocation matrix values for each process zero and also the values in available vector are set to the number of instance of each resource respectively and the maximum matrix is set randomly, The program calculate the need matrix by subtracting the allocation matrix from maximum matrix and generate a request for each process randomly, and checks the safe state as

1. We have Work and Finish to be vectors of length of number of processes and number of resources, respectively.

By Initializing:

Work = Available

Finish [i] = false for i = 0, 1,…, n- 1

1. Find an i such that both:

(a) Finish [i] = false

(b) Need of (i) ≤ Work

If no such i exists, go to step 4

1. Work = Work + Allocation of (i)

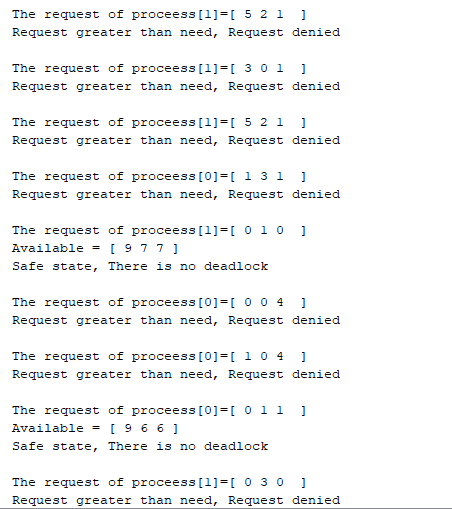
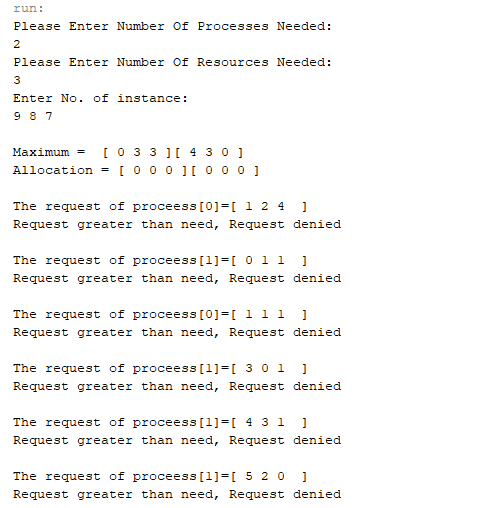
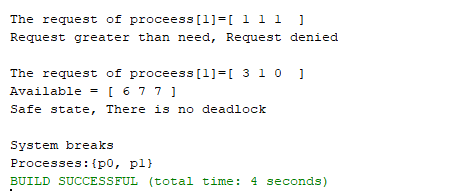
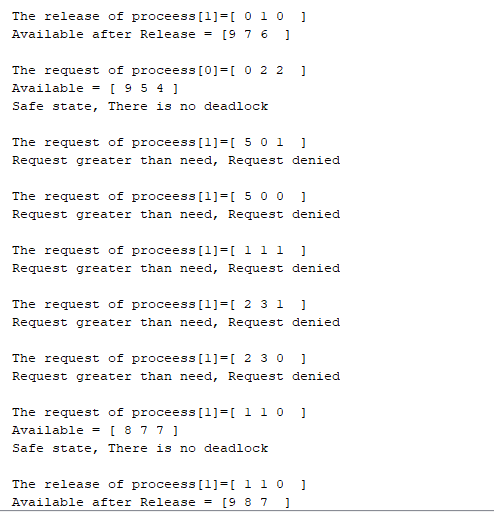
Finish[i] = true

Go to step 2

1. If Finish [i] == true for all i, then the system is in a safe state

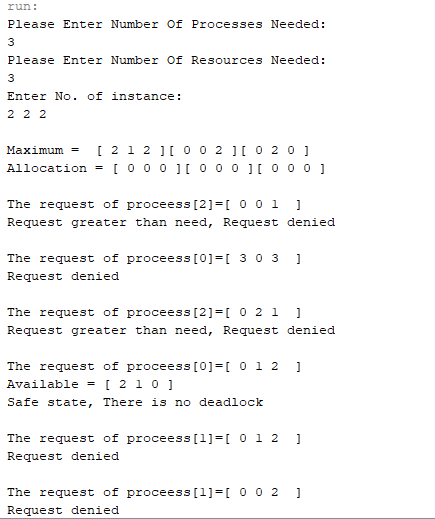
If it is true the request is accepted and change the values of the matrices as Available = Available – Request of the selected process; Allocation of the selected process = Allocation of the selected process + Request of the selected process; Need of the selected process = Need of the selected process – Request of the selected process;, But if the check is false the request is denied, and the values of the matrices don’t change and the program generate a new request. The program can generate a random index of process to release a random number of its allocation to be added into the available vector.The program ends when the need values is zero for all processes. Noting that the program doesn’t accept a request that is greater than the need of the selected process and generate a new request.

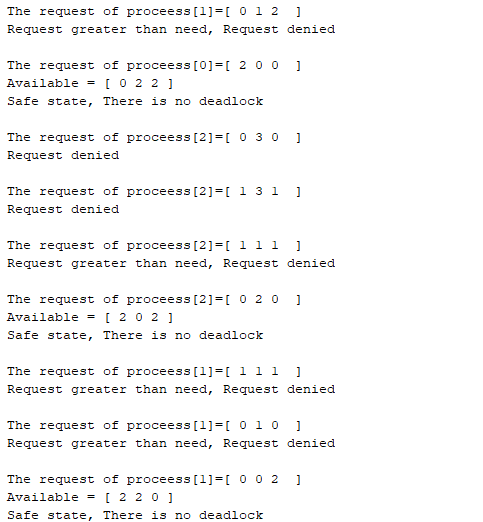
**Test Cases:**

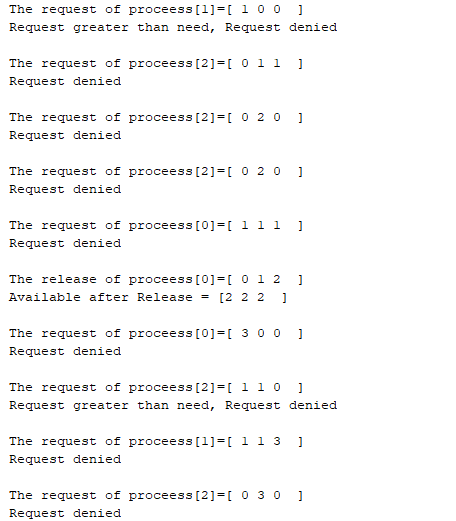
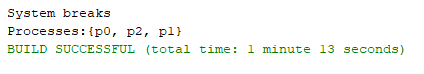
****Test case 1:

Comment: Number of process less than the number

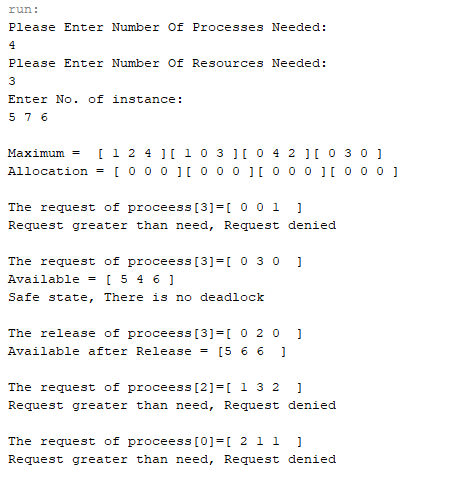
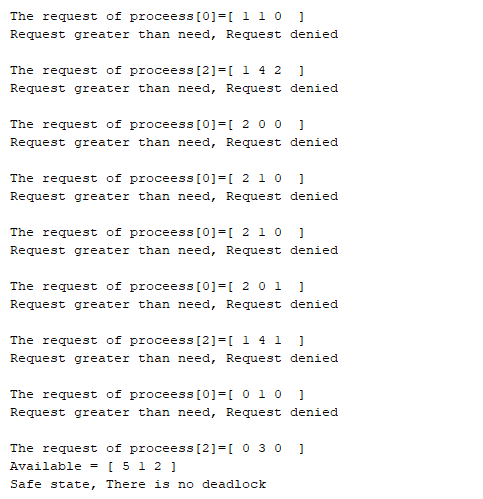
of resources.

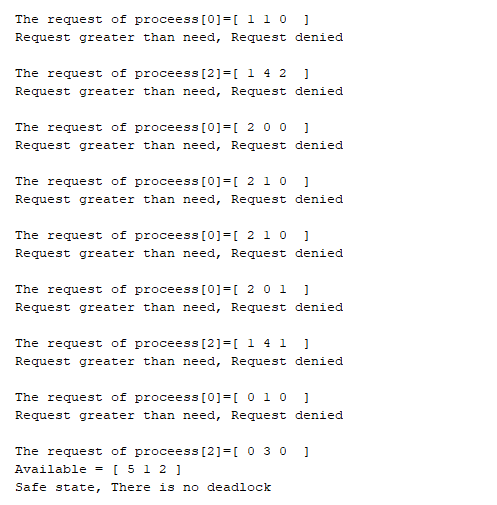
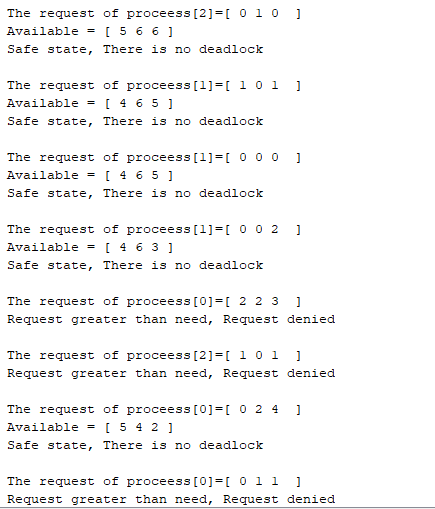
Test case 2:

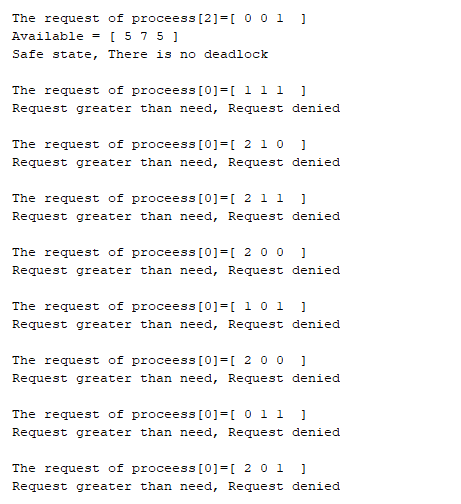
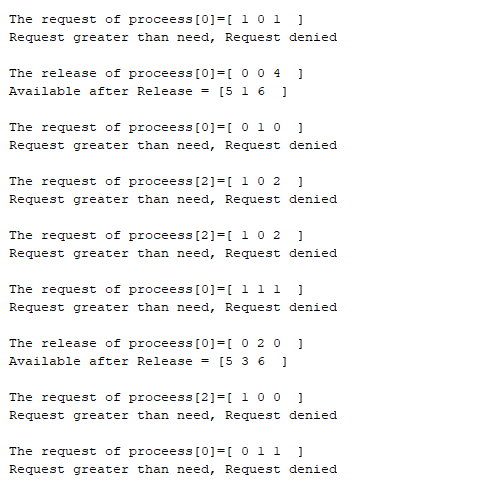
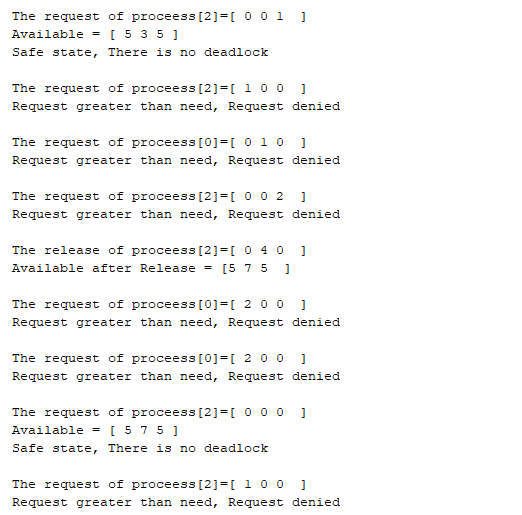
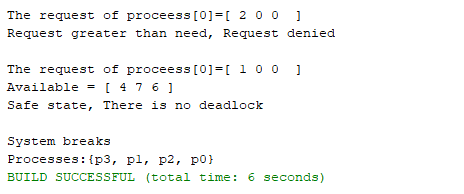
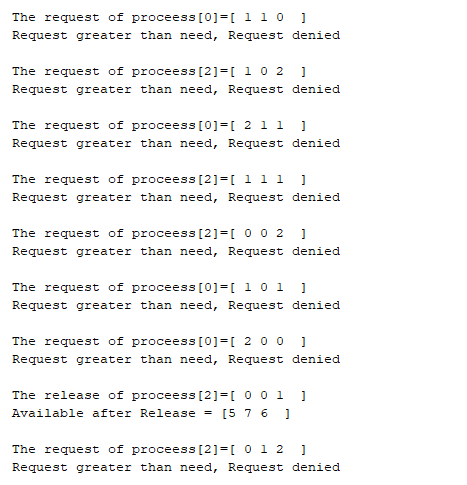


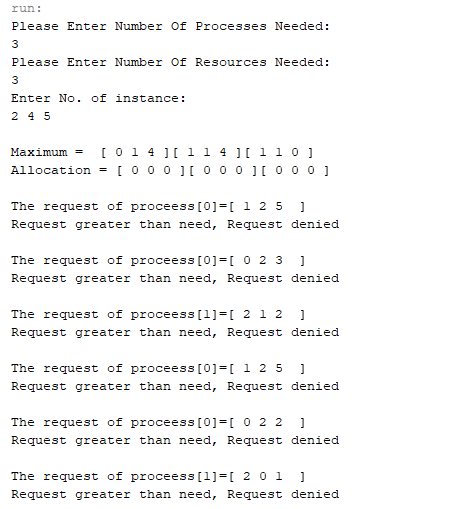


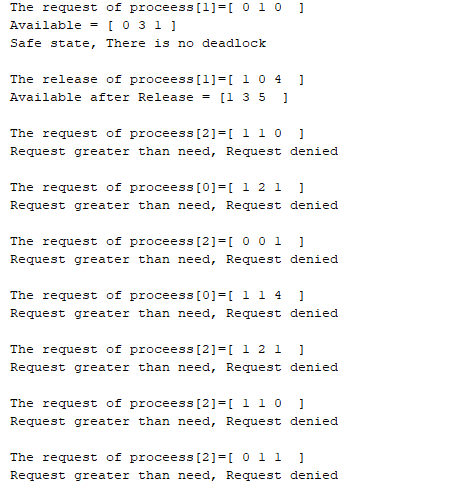
Comment: Number of process equal Number of resources. Number of instance is relatively small.

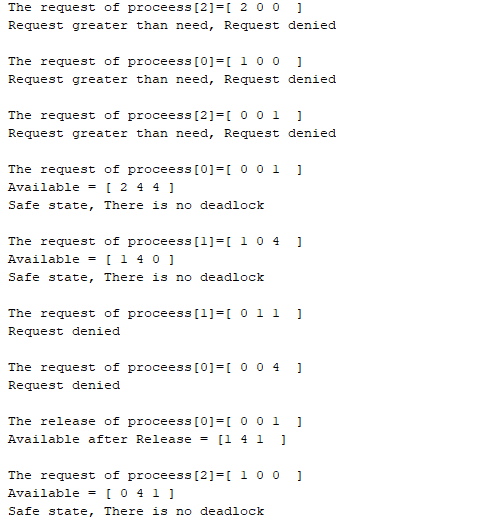
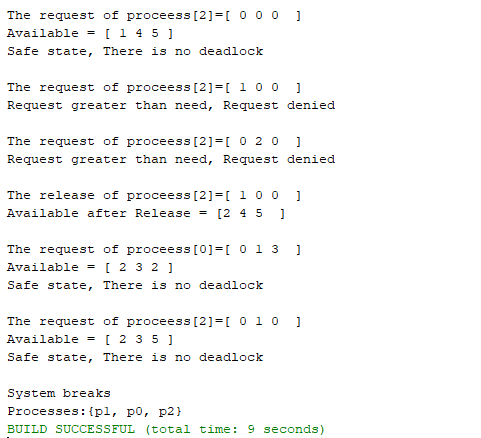
Test Case 3:



 Comment: the number of process is greater than the number of resources.

Test case 4:

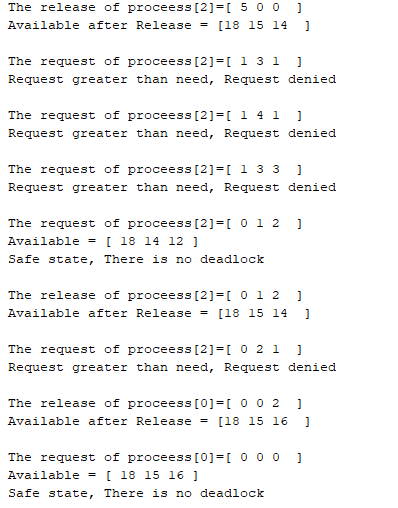
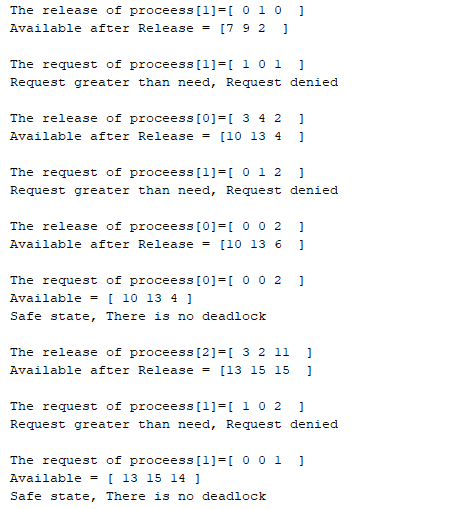
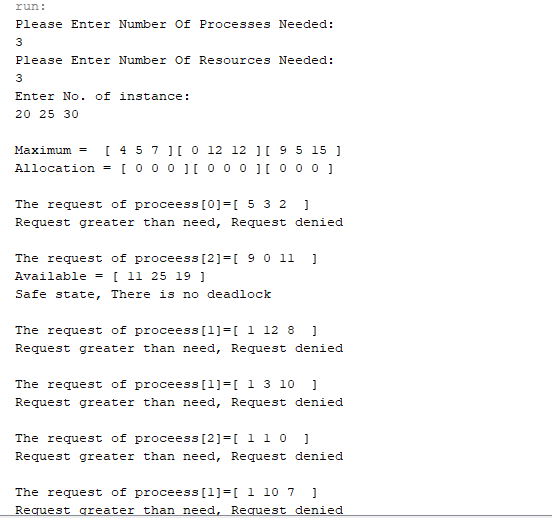
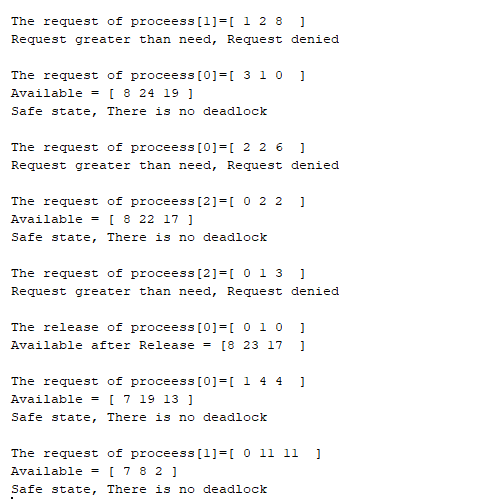


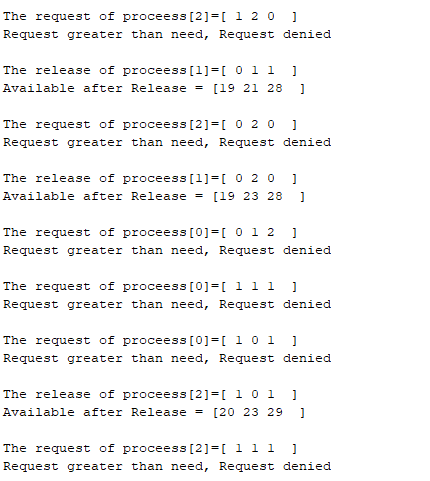
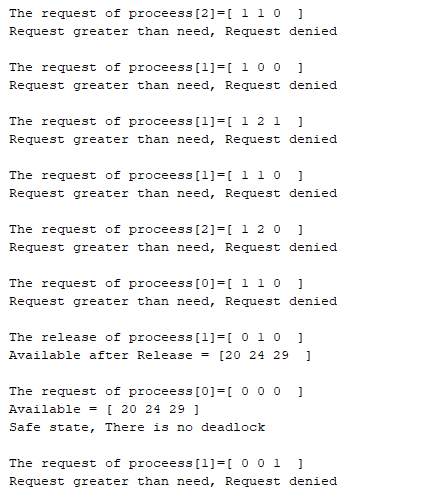
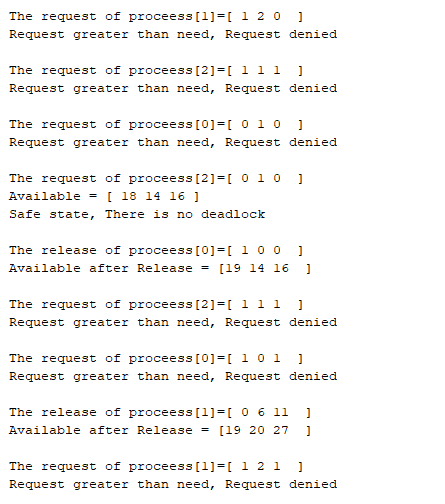
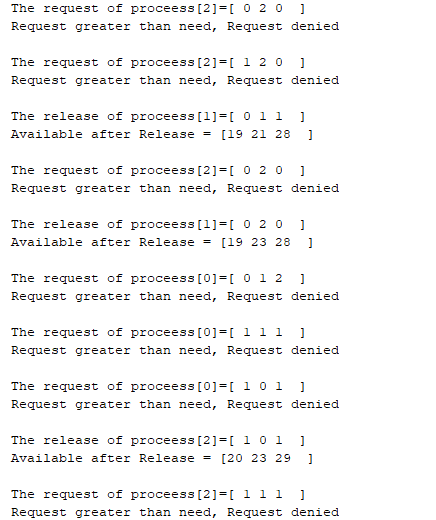


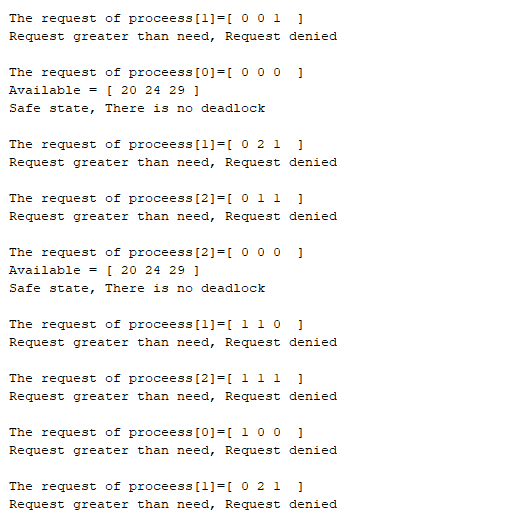
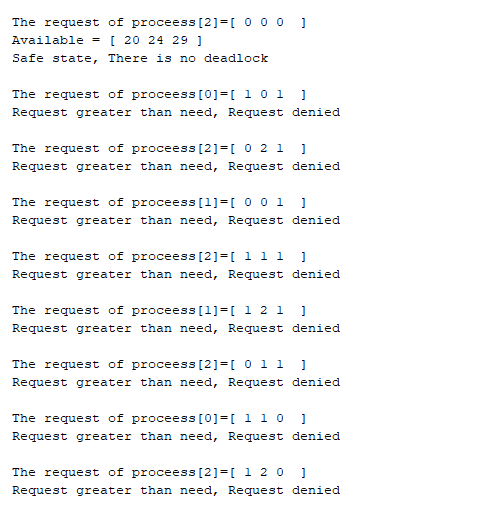
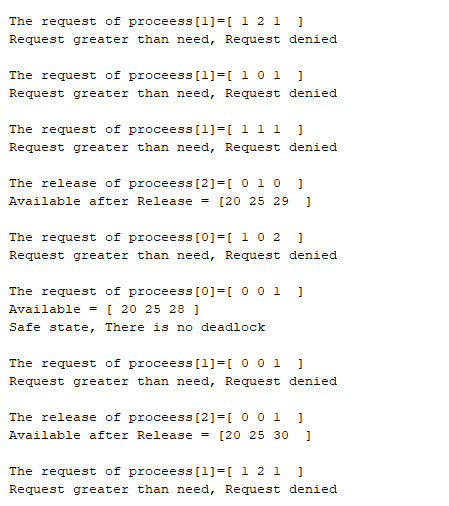
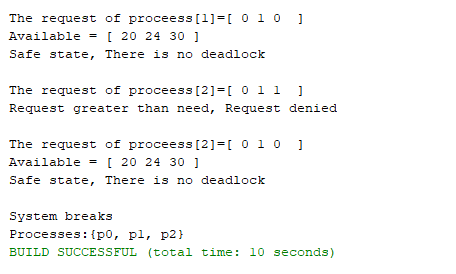
Comment: In this case the program generate a

request greater than the available vector so the

program denied this request and generate a new request.

Test case 5:





Comment: In this case the number of

instance is so large.