CSE 310: Operating System Lab

Lab 08

Deadlock Management

Task 1: Resource Allocation Graph (RAG)

Objective: Deadlock detection using a resource allocation graph for a single quantity of each resource using a cycle detection algorithm.

IDE: C/C++ , Java

Background

Deadlock:

From the application point of view, a deadlock is a situation in which two computer programs are sharing the same resource and effectively preventing each other from accessing the resource, resulting in both programs ceasing to function.

Resource Allocation Graph (RAG)

In the Resource Allocation Graph , Vertices (Rectangle : Resource or Circle: process) represent process and resources. The directed edge from a process P to a resource R represents that the process P is willing to get resource R. The directed edge from a resource R to a process P represents that the process P is holding the resource R in its custody.

Deadlock Detection using Resource Allocation Graph (RAG)

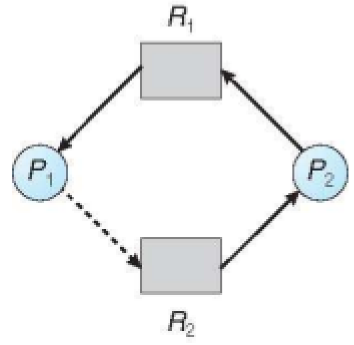
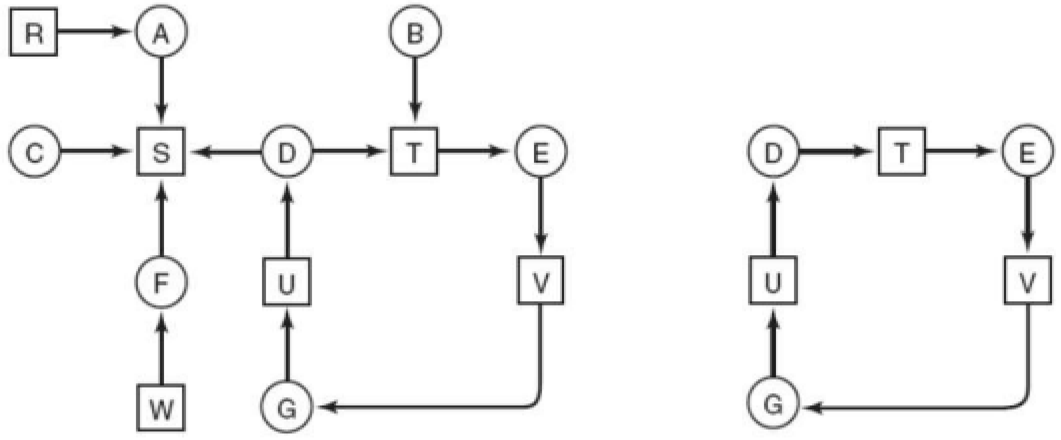


Fig1: A simple RAG

If there is a cycle in the Resource Allocation Graph and each resource in the cycle provides only one instance, then the processes will be in deadlock. For example, if process P1 holds resource

R1, process P2 holds resource R2 and process P1 is waiting for R2 and process P2 is waiting for R1, then process P1 and process P2 will be in deadlock.

Deadlock detection using a Cycle detection method



Algorithm

1. For each node, N in the graph, perform the following five steps with N as the starting node.
2. Initialize L to the empty list, designate all arcs as unmarked.
3. Add current node to end of L, check to see if node now appears in L two times. If it does, graph contains a cycle (listed in L), algorithm terminates.
4. From given node, see if any unmarked outgoing arcs. If so, go to step 5; if not, go to step 6.
5. Pick an unmarked outgoing arc at random and mark it. Then follow it to the new current node and go to step 3.
6. If this is initial node, graph does not contain any cycles, algorithm terminates. Otherwise, dead end. Remove it, go back to previous node, make that one current node, go to step 3.

Input	**Output**
Number of Nodes: 13 Number of Edges: 13 Edges: R A A S C S F S W F D S U D G U V G D T T E E V B T	Deadlock: Yes

Task 2: Banker’s Algorithm

Objective: Simulate Banker’s Algorithm used for Deadlock Avoidance and find whether the system is in safe state or not.
IDE: C/C++ , Java

Background :
In Banker’s algorithm, when a new process enters a system, it declares the maximum number of instances of each resource type it needed. This number may exceed the total number of resources currently available in the system. When a process requests a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will, the resources are allocated; otherwise the process must wait until some other process release the resources. If all the processes in the system can be satisfied in any order
by giving them their requested number of resources, the system is said to be in safe state and the order of serving is called the safe sequence.

Working Procedure
Let us assume that there are n processes and m resource types. Some data structures that are used to implement the banker's algorithm are:
1. Available
It is an array of length m. It represents the number of available resources of each type. If Available[j] = k, then there are k instances available, of resource type R(j).
2. Max
It is an n x m matrix which represents the maximum number of instances of each resourcethat a process can request. If Max[i][j] = k, then the process P(i) can request atmost k instances of resource type R(j).
3. Allocation
It is an n x m matrix which represents the number of resources of each type currently allocated to each process. If Allocation[i][j] = k, then process P(i) is currently allocated k instances of resource type R(j).
4. Need
It is an n x m matrix which indicates the remaining resource needs of each process. If Need[i][j] = k, then process P(i) may need k more instances of resource type R(j) to complete its task. Here , Need[i][j] = Max[i][j] - Allocation [i][j]

Safety Algorithm
1. Let Work and Finish be vectors of length m and n, respectively. Initially,
Work = Available
Finish[i] =false for i = 0, 1, ... , n - 1.

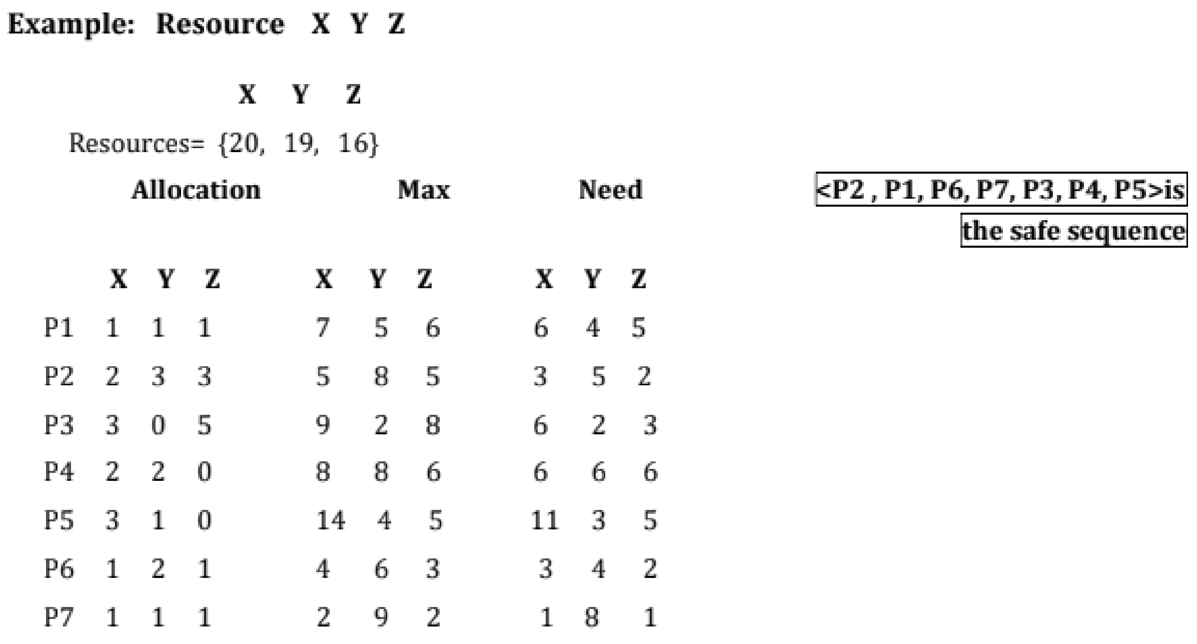
This means, initially, no process has finished and the number of available resources is represented by the Available array.

2. Find an index i such that both Finish[i] ==false and Needi <= Work
If there is no such i present, then proceed to step 4.

It means, we need to find an unfinished process whose need can be satisfied by the available resources. If no such process exists, just go to step 4.

3. Perform the following:
Work = Work + Allocation;
Finish[i] = true;
Go to step 2.
When an unfinished process is found, then the resources are allocated and the process is marked finished. And then, the loop is repeated to check the same for all other processes.

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



Example: Resource X Y Z

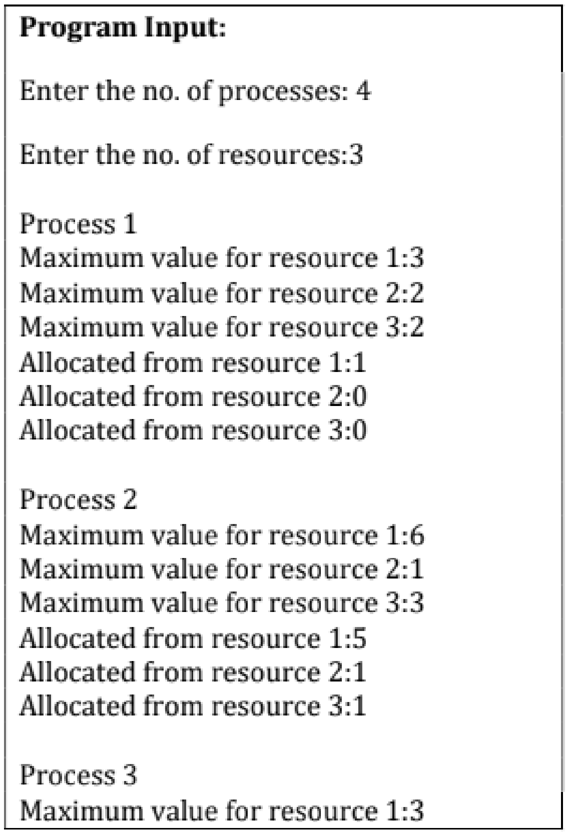
X Y Z

Resources= {20, 19, 16}

	Allocation			**Max**			**Need**		
	X	**Y**	**Z**	**X**	**Y**	**Z**	**X**	**Y**	**Z**
P1	1	1	1	7	5	6	6	4	5
P2	2	3	3	5	8	5	3	5	2
P3	3	0	5	9	2	8	6	2	3
P4	2	2	0	8	8	6	6	6	6
P5	3	1	0	14	4	5	11	3	5
P6	1	2	1	4	6	3	3	4	2
P7	1	1	1	2	9	2	1	8	1

<P2 , P1, P6, P7, P3, P4, P5>is the safe sequence

Task: Given a set of data for the Banker's algorithm, detect whether the system is in safe state or not



Program Input:

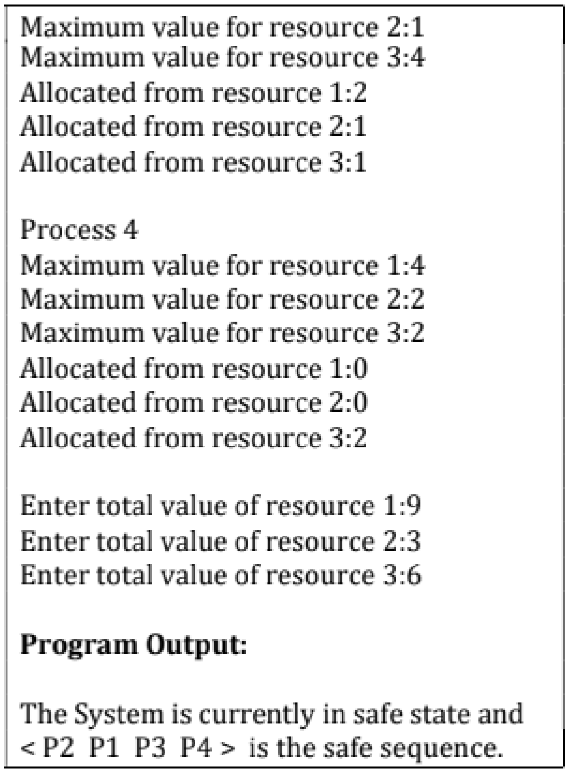
```
Enter the no. of processes: 4

Enter the no. of resources:3

Process 1
Maximum value for resource 1:3
Maximum value for resource 2:2
Maximum value for resource 3:2
Allocated from resource 1:1
Allocated from resource 2:0
Allocated from resource 3:0

Process 2
Maximum value for resource 1:6
Maximum value for resource 2:1
Maximum value for resource 3:3
Allocated from resource 1:5
Allocated from resource 2:1
Allocated from resource 3:1

Process 3
Maximum value for resource 1:3
```



```
Maximum value for resource 2:1
Maximum value for resource 3:4
Allocated from resource 1:2
Allocated from resource 2:1
Allocated from resource 3:1

Process 4
Maximum value for resource 1:4
Maximum value for resource 2:2
Maximum value for resource 3:2
Allocated from resource 1:0
Allocated from resource 2:0
Allocated from resource 3:2

Enter total value of resource 1:9
Enter total value of resource 2:3
Enter total value of resource 3:6
```

Program Output:

```
The System is currently in safe state and
< P2  P1  P3  P4 >  is the safe sequence.
```