ASSIGNMENT NO.: 05

AIM:

Implement the C program for Deadlock Avoidance Algorithm: Bankers Algorithm.

PREREQUISITE:

1. C Programming
2. Fundamentals of Data Structure

OBJECTIVE:

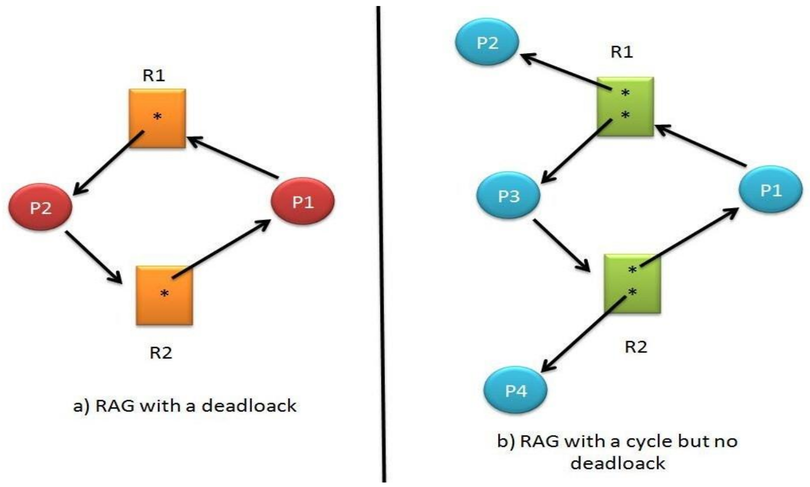
To Study

- Deadlock
- Deadlock Avoidance Algorithm- Bankers Algorithm.

THEORY:

What is Deadlock?

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause. Because all the processes are waiting, none of them will ever cause any of the events that could wake up any of the other members of the set, and all the processes continue to wait forever. For this model, we assume that processes have only a single thread and that there are no interrupts possible to wake up a blocked process. The no interrupts condition is needed to prevent an otherwise deadlocked process from being awake.



a) RAG with a deadloack

b) RAG with a cycle but no deadloack

Conditions for Deadlock

Coffman et al. (1971) showed that four conditions must hold for there to be a deadlock:

- ❖ Mutual exclusion condition- Each resource is either currently assigned to exactly one process or is available.
- ❖ Hold and wait condition- Processes currently holding resources granted earlier can request new resources.
- ❖ No preemption condition- Resources previously granted cannot be forcibly taken away from a process. The process holding them must explicitly release them.
- ❖ Circular wait condition- There must be a circular chain of two or more processes, each of which is waiting for a resource held by the next member of the chain.

Banker's Algorithm in Operating System

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.

Why Banker's algorithm is named so?

Banker's algorithm is named so because it is used in banking system to check whether loan can be sanctioned to a person or not. Suppose there are ***n*** number of account holders in a bank and the total sum of their money is ***S***. If a person applies for a loan then the bank first subtracts the loan amount from the total money that bank has and if the remaining amount is greater than S then only the loan is sanctioned. It is done because if all the account holders comes to withdraw their money then the bank can easily do it.

In other words, the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers. The bank would try to be in safe state always.

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 $\mathbf{R_j}$

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 $\mathbf{P_i}$ may request at most **'k'** instances of resource type $\mathbf{R_j}$.

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 $\mathbf{P_i}$ is currently allocated **'k'** instances of resource type $\mathbf{R_j}$

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 $\mathbf{P_i}$ currently need **'k'** instances of resource type $\mathbf{R_j}$
- Need [i, j] = Max [i, j] – Allocation [i, j]

Allocation specifies the resources currently allocated to process P_i and Need specifies the additional resources that process P_i 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

Resource-Request Algorithm

Let $Request_i$ be the request array for process P_i. $Request_i$ [j] = k means process P_i wants k instances of resource type R_j. When a request for resources is made by process P_i, the following actions are taken:

1) If $Request_i$ <= $Need_i$

Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

2) If $Request_i$ <= Available

Goto step (3); otherwise, P_i must wait, since the resources are not available.

3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:

Available = Available – Requesti

$Allocation_i = Allocation_i + Request_i$

$Need_i = Need_i - Request_i$

Example-

Considering a system with five processes P0 through P4 and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t0 following snapshot of the system has been taken:

Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

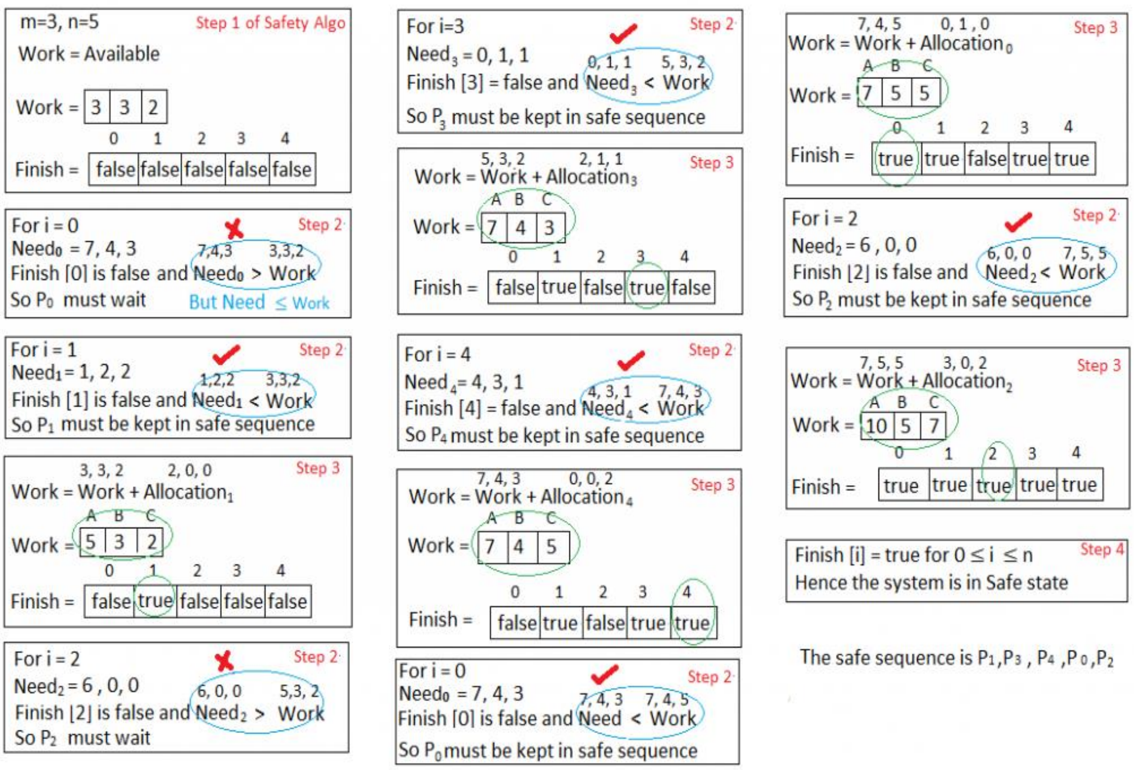
Question1. What will be the content of the Need matrix?

Need [i, j] = Max [i, j] – Allocation [i, j]

So, the content of Need Matrix is:

Process	Need		
	A	B	C
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

Applying the Safety algorithm on the given system,



The safe sequence is P_1, P_3, P_4, P_0, P_2

What will happen if process P1 requests one additional instance of resource type A and two instances of resource type C

A B C
$Request_1$ = 1, 0, 2

To decide whether the request is granted we use Resource Request algorithm

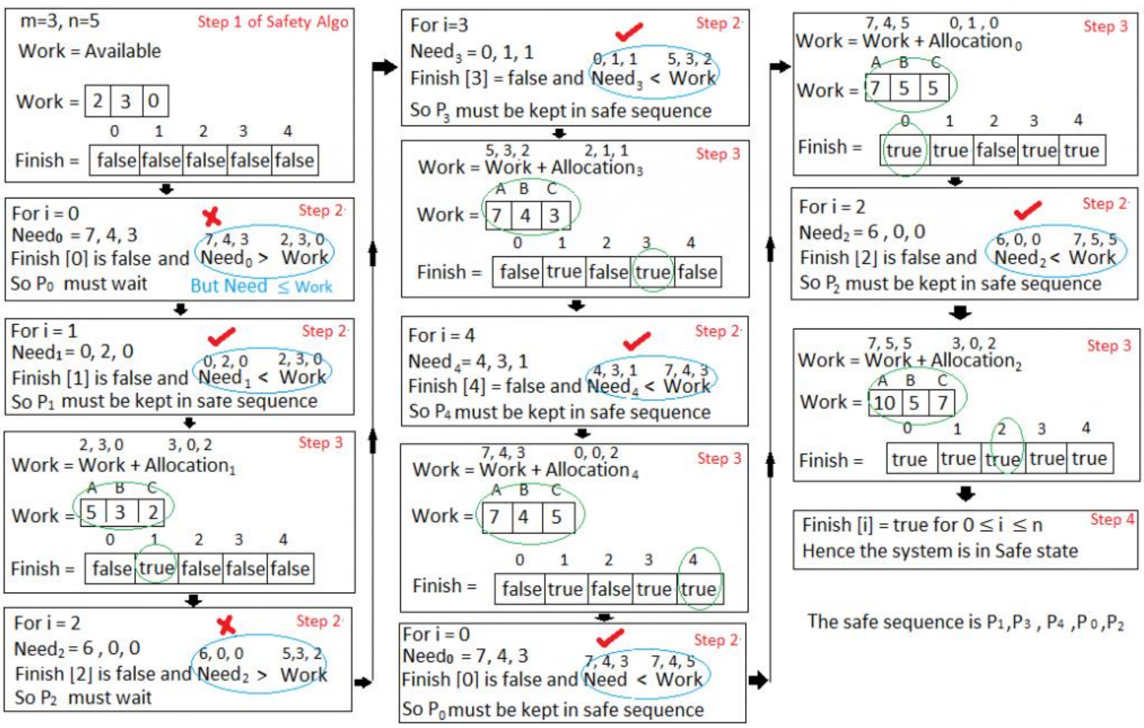
Step 1
1, 0, 2 1, 2, 2
$Request_1 < Need_1$

Step 2
1, 0, 2 3, 3, 2
$Request_1 < Available$

Step 3
Available = Available – $Request_1$
$Allocation_1$ = $Allocation_1$ + $Request_1$
$Need_1$ = $Need_1$ - $Request_1$

Process	Allocation	Need	Available
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

We must determine whether this new system state is safe. To do so, we again execute Safety algorithm on the above data structures.



CONCLUSION:

Thus, we have implemented Banker's Algorithm problem using 'C' in Linux.

FAQ:

1. What is dead lock?
2. What are the necessary and sufficient conditions to occur deadlock?
3. What is deadlock avoidance and deadlock prevention techniques?

OUTPUT:

(Attach Screenshots of your output in sequence)