Assignment Number: 05

TITLE: Bankers Algorithm.

Problem statement- Implement the C program for Deadlock Avoidance Algorithm: Bankers Algorithm

OBJECTIVE:

1. To study deadlock situation in operating system.
2. To understand Bankers algorithm for deadlock avoidance and detection.

Theory:

An approach to solving the deadlock problem that differs subtly from deadlock prevention is deadlock avoidance. In deadlock prevention, we constrain resource requests to prevent at least one of the four conditions of deadlock. This is either done indirectly, by preventing one of the three necessary policy conditions (mutual exclusion, hold and wait, no preemption), or directly by preventing circular wait. This leads to inefficient use of resources and inefficient execution of processes. Deadlock avoidance, on the other hand, allows the three necessary conditions but makes judicious choices to assure that the deadlock point is never reached. As such, avoidance allows more concurrency than prevention. With deadlock avoidance, a decision is made dynamically whether the current resource allocation request will, if granted, potentially lead to a deadlock. Deadlock avoidance thus requires knowledge of future process resource requests.
We describe two approaches to deadlock avoidance:

- Do not start a process if its demands might lead to deadlock.
- Do not grant an incremental resource request to a process if this allocation might lead to deadlock.

Resource Allocation Denial:

Consider a system of n processes and m different types of resources. Let us define the following vectors and matrices:

[Resource = $\mathbf{R} = (R_1, R_2, \ldots, R_m)$	total amount of each resource in the system
Available = $\mathbf{V} = (V_1, V_2, \ldots, V_m)$	total amount of each resource not allocated to any process
Claim = $\mathbf{C} = \begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1m} \\ C_{21} & C_{22} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ C_{n1} & C_{n2} & \cdots & C_{nm} \end{bmatrix}$	C_{ij} = requirement of process i for resource j
Allocation = $\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{bmatrix}$	A_{ij} = current allocation to process i of resource j

The matrix Claim gives the maximum requirement of each process for each resource, with one row dedicated to each process. This information must be declared in advance by a process for deadlock avoidance to work. Similarly, the matrix Allocation gives the current allocation to each process. The following relationships hold:

$R_j = V_j + \sum_{i=1}^{n} A_{ij}$, for all j	All resources are either available or allocated.
$C_{ij} \leq R_j$, for all i, j	No process can claim more than the total amount of resources in the system.
$A_{ij} \leq C_{ij}$, for all i, j	No process is allocated more resources of any type than the process originally claimed to need.

With these quantities defined, we can define a deadlock avoidance policy that refuses to start a new process if its resource requirements might lead to deadlock. Start a new process Pn+1 only if

$$R_j \geq C_{(n+1)j} + \sum_{i=1}^{n} C_{ij} \quad \text{for all } j$$

That is, a process is only started if the maximum claim of all current processes plus those of the new process can be met. This strategy is hardly optimal, because it assumes the worst: that all processes will make their maximum claims together.

Resource Allocation Denial

The strategy of resource allocation denial, referred to as the banker's algorithm, was first proposed by Edsger Dijkstra. Let us begin by defining the concepts of state and safe state. Consider a system with a fixed number of processes and a fixed number of resources. At any time a process may have zero or more resources allocated to it. The state of the system reflects the current allocation of resources to processes. Thus, the state consists of the two vectors, Resource and Available, and the two matrices, Claim and Allocation, defined earlier. A safe state is one in which there is at least one sequence of resource allocations to processes that does not result in a deadlock (i.e., all of the processes can be run to completion). An unsafe state is, of course, a state that is not safe.

Example of finding the sequence and the safe state:

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix C

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Allocation matrix A

	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector R

R1	R2	R3
0	1	1

Available vector V

(a) Initial state

	R1	R2	R3
P1	3	2	2
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim matrix C

	R1	R2	R3
P1	1	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation matrix A

	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector R

R1	R2	R3
6	2	3

Available vector V

(b) P2 runs to completion

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim matrix C

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation matrix A

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector R

R1	R2	R3
7	2	3

Available vector V

(c) P1 runs to completion

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	2

Claim matrix C

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	2

Allocation matrix A

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector R

R1	R2	R3
9	3	4

Available vector V

(d) P3 runs to completion

Conclusion
Write in your own words

References:

1. Operating System design and Internals by William Stallings, Prentice Hall of India.
2. Operating System Concepts by Galvin, Silberchartz, Tata McGraw Hill.