*Deadlock Avoidance

Lab 3

*BACKGROUND

- *In deadlock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system.
- * The state of the system will continuously be checked for safe and unsafe states.
- *In order to avoid deadlocks, the process must tell os, the maximum number of resources a process can request to complete its execution.
- *The deadlock avoidance algorithm examines the resource allocations so that there can never be a circular wait condition.

*Definitions

- *In **deadlock avoidance** Each process provides OS with information about its requests and releases for resources ri
- *OS decides whether deadlock will occur at run time
- *e.g.OS jobs know a priori which resources they’ll request and when
- * weakness : need a priori info
- *knowing all future requests and releases is difficult
- * simple strategy: specify a maximum claim

• estimating a maximum demand for resources for each process is easier to produce

*What is the state?

* The state of the system informs that if resources are allocated to different processes then the system under-goes deadlock or not.

Safe and Unsafe States

- If the system can allocate resources to the process (up to their stated maximums) in such a way that it can avoid deadlock. Then the system is in a safe state.
- A safe state provides a safe “escape” sequence
- If the system can’t allocate resources to the process safely, then the system is in an unsafe state.
- **Note:** *unsafe state not always cause deadlock*.
- A deadlocked state is unsafe
- A system may transition from a safe to an unsafe state
- ideally, check with each request for resources whether the system is still safe

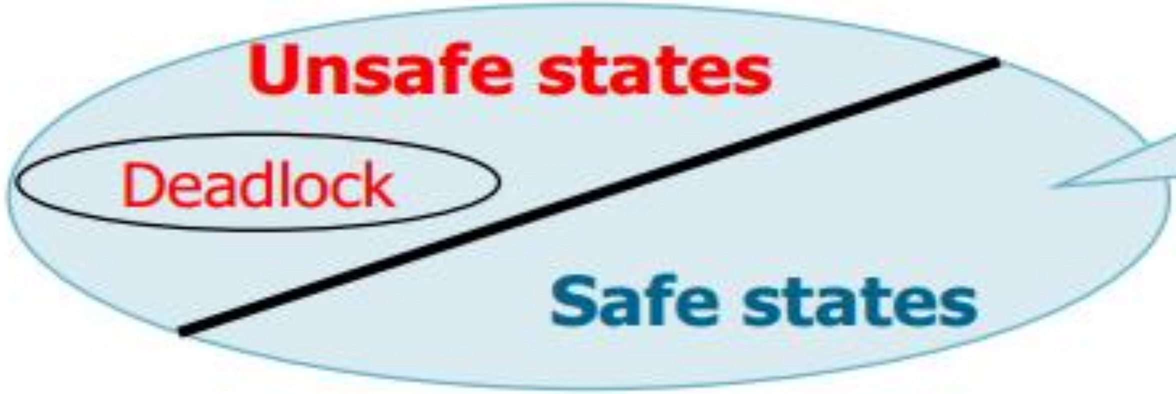
*More details:

* If a safe sequence does not exist, then the system is in an unsafe state, which MAY lead to deadlock.
* System is in safe state if there exists a sequence <p1, p2, …, pn> of all the processes in the systems such that for each pi.

* the resources that pi can still request can be satisfied by: currently available resources + resources held by all the pj, with j < i

- **Then:**
- If, pi resource does not require immediately available, then pi have to wait until completely pj have been finished.
- Whenever pj is getting ending, then pi can receive required resources, execute, return allocated resources, and terminate.
- If pi terminates, then pi+1 can get its required resources, and so on.

Safe state	The system is able to • Allocate the required resources to all processes • Prevent the occurrence of a deadlock • Find a safe sequence
Safe sequence	A sequence of process scheduling $\{P_1, P_2, \ldots, P_n\}$ such that for each requests that could be performed by any P_i, the request can be satisfied by using the currently available resources and the other resources released by processes P_j with $j < i$



A state is said **unsafe** if it is not safe.
An unsafe state is not necessarily a deadlock state. It can leads to a deadlock state in case of standard behavior.

*Primary Facts –

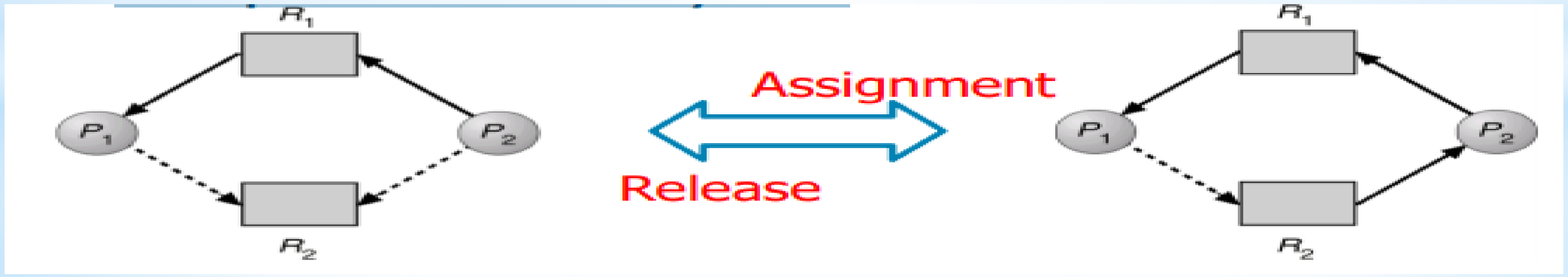
- Initially the system is in a safe state
- If system is getting in the safe state, then does not happen deadlock
- If system is getting in the unsafe state, then may be occur deadlock
- If ensure that system will never going to enter an unsafe state, then avoidance is getting.

*Avoidance algorithms

- *Single instance of a resource type use a resource-allocation graph.
- *multiple instances of a resource type use the banker's algorithm.

Algorithm for resources with a single instance

- Based on the determination of cycles, using the claim-for graph
  - All requests must be a priori declared
  - they are represented by claim arcs
- At a time a request is performed
  - the corresponding claim arc is transformed into an assignment arc
  - Before the request is satisfied, the algorithm verify the presence of cycles



*Banker's Algorithm (Dijkstra, [1965])

- Bankers's algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it checks for the safe state.
- if after granting request system remains in the safe state it allows the request and if there is no safe state it doesn't allow the request made by the process.
- Multiple instances each process must a priori claim maximum use
- when a process requests a resource it may have to wait, when a process gets all its resources it must return them in a finite amount of time.

* Data Structures used to implement the Banker's Algorithm

1. Available
2. Max
3. Allocation
4. Need

1. Available

- Available is A one-dimensional array. The size of the array is ‘M’ which is used to determine the number of available resources of each kind.
- Available[j] = k indicates that we have ‘k’ instances of ‘rj’ resource type.

2. Max

- Max is A two-dimensional array. The size of the array is ‘n*m’. The max data structure is used to determine the maximum number of resources that each process requests.
- Max[i, j] = k indicates that ‘pi’ can demand or request maximum ‘k’ instances of ‘rj’ resource type.

3. Allocation

- Allocation is A two-dimensional array, and the size of the array is ‘n*m’, which is used to define the number of resources of each kind presently assigned to each process.
- Allocation[i, j] = k indicates that currently process ‘pi’ is assigned ‘k’ instances of ‘rj’ resource type.

4. Need

- Need is A two-dimensional array. The size of the array is ‘n*m’. Need is used to define the remaining resources which are required for each process.
- Need [i, j] = k indicates that for the execution of ‘pi’ process, presently ‘k’ instances of resource type ‘Rj’ are required.

*Inputs to Banker’s Algorithm:

1. Max need of resources by each process.
2. Currently, allocated resources by each process.
3. Max free available resources in the system.

*The request will only be granted under the below condition:

1. If the request made by the process is less than equal to max need to that process.
2. If the request made by the process is less than equal to the freely available resource in the system.
3. And the rule will be:

   Need = maximum resources - currently allocated resources.

Example of Banker's Algorithm

- 5 processes P_0 through P_4; 3 resource types *A* (10 instances), *B* (5instances, and *C* (7 instances).
- Snapshot at time T_0:

	Allocation	*Max*	*Need*	*Total*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	7 4 3	10 5 7
P_1	2 0 0	3 2 2	1 2 2	*Allocated*
P_2	3 0 2	9 0 2	6 0 0	7 2 5
P_3	2 1 1	2 2 2	0 1 1	*Available*
P_4	0 0 2	4 3 3	4 3 1	3 3 2

- The system is in a safe state since the sequence $< P_1, P_3, P_4, P_2, P_0>$ satisfies safety criteria

	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0	0	0	0	0
P1	1	0	0	0	1	7	5	0	1	5	3	2	0	7	5	0
P2	1	3	5	4	2	3	5	6	2	8	8	6	1	0	0	2
P3	0	6	3	2	0	6	5	2	2	14	11	8	0	0	2	0
P4	0	0	1	4	0	6	5	6	2	14	12	12	0	6	4	2
									3	14	12	12				

Safe Sequence : P0 P2 P3 P4 P1

*Disadvantages of Banker’s Algorithm

1. During processing, it does not permit a process to change its maximum need.
2. All the processes should know in advance about the maximum resource needs.
3. Banker’s algorithm permits the requests to be provided in constrained time

Homework

Write a Method in C# to calculate the “Need”.