Tutorial 3: Deadlock

Q1. (a) Discuss how a deadlock situation can arise if the **FOUR (4)** deadlock conditions hold simultaneously in a multiprogramming environment. Recommend a solution to solve each deadlock condition.

Mutual exclusion

Situation

- the act of allowing only one process to have access to a dedicated resource
- At least one resource must be held in a non-shareable mode. Otherwise, the processes would not be prevented from using the resource when necessary. Only one process can use the resource at any given instant of time

Solution – Must hold for non-sharable resource; not required for mutually exclusive access. (E.g., Read only files)

Resource Holding

Situation - This happen when a job is holding a resource and the other job is waiting for the job to release the resource

Solution

– Must guarantee that whenever a process requests a resource, it does not hold any other resources

- Allocate every necessary resource to the job at creation time so that the job can be completed when the needed resource is not held by other job.

No pre-emption

Situation – The lack of temporary reallocation of resources; once a job gets a resource It can hold on to it as long as it needs

Solution

– Could be bypasses by allowing OS to deallocate resources from jobs

Circular wait

Situation – Each process involved in an impasse is waiting for another to voluntarily release the resource so that at least one will be able to continue

Solution

– Prevent circle formation by using hierarchical ordering scheme

- Impose a total ordering of all resource types, and require that each process requests resources in an increasing order or enumeration

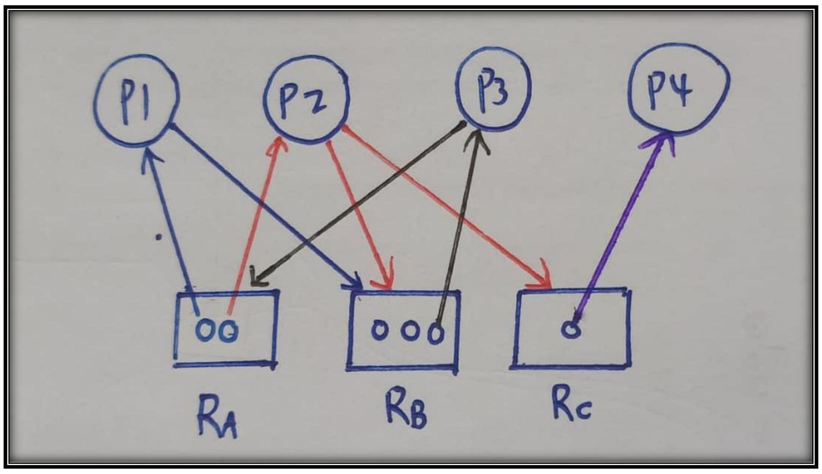
(b) Propose THREE (3) possible ways that an operating system (OS) can handle the problem of deadlock.

- Prevent one of the four deadlock conditions from occurring
- Avoid the deadlock if it becomes probable
- Detect the deadlock when it occurs and recover from it gracefully

Q2. (a) Consider a system with four processes P1, P2, P3 and P4 and three resources types A, B and C. Resource type A has 2 instances, resource type B has 3 instances and resource type C has 1 instance.

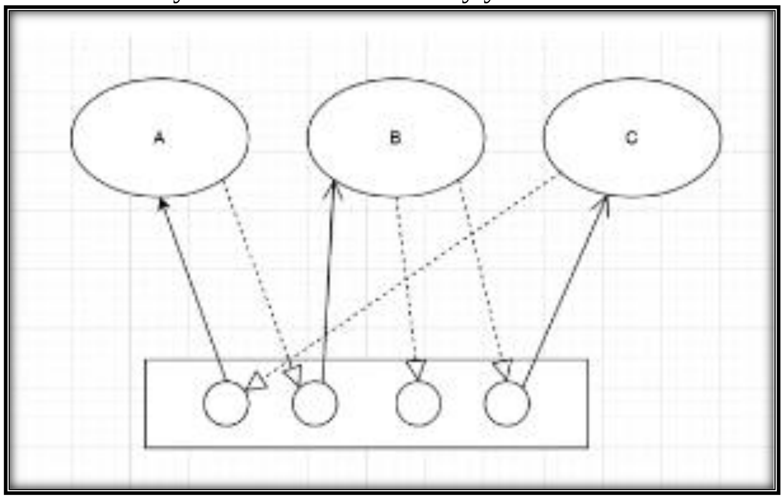
- P1 holds an instance of A and requests an instance of B.
- P2 holds an instance of A and requests an instance of B and C.
- P3 holds an instance of B and requests an instance of A.
- P4 holds an instance of C.

Draw a Directed Resources Graph for the above situation. *Request process use dot line



(b) Are these four processes mentioned in Q3. (a) in a deadlocked state? Justify your answer.
P1, P3, P4, P2
This is not a deadlock, after P1 is executed, one instance of Resource A is released, P2 will have enough resource to be execute after P4 release the instance in Resource C. P3 is able to be execute after the instance of Resource A holding by P1 is released

Q3. Consider a system consisting of 4 resources of the same type that are shared by 3 processes, each of which needs at most 2 resources. You are required to draw a resource-allocation graph to show that the system is deadlock. Justify your answer



System is in deadlock, Circular wait occurs. Process A will have to wait for resource that is hold by Process B to be released in order to execute while process B have to wait for resource that is hold by Process A to be release in order to execute, therefore circular wait occurs.

Q4. Dijkstra's algorithm is a deadlock avoidance algorithm and can be used to prevent the occurrence of deadlock. Suppose at time, t_j, a snapshot of the system is taken and described as shown as Table 1.

Processes	**Allocated Resources** **X Y Z**	**Maximum Requirements** **X Y Z**
P_a	1 1 0	5 6 4
P_b	2 2 3	2 4 5
P_c	4 2 0	7 3 1

Table 1: System snapshot at time, t_j

(i) Calculate the content of the matrix *Need*.

Need = Maximum Requirements – Allocated Resources

Processes	**Need** **X Y Z**
P_a	4 5 4
P_b	0 2 2
P_c	3 1 1

(ii) Assuming that the available resources for X, Y and Z are 3, 1, and 2 respectively, what is the total number of resource instances for resource X, Y and Z?

- Total number of resources = available + allocated

- X = 10, Y = 6, Z = 5

(iii) Is the system in a safe state? If yes state the process sequence, if no identify the deadlock processes. Justify your answer by showing the progress of resources availability.

Processes	**Allocated Resources** **X Y Z**	**Maximum Requirements** **X Y Z**	**Need** **X Y Z**	**Available** **X Y Z**
				3 1 2
P_c	4 2 0	7 3 1	3 1 1	7 3 2
P_b	2 2 3	2 4 5	0 2 2	9 5 5
P_a	1 1 0	5 6 4	4 5 4	**10 6 5**

- The system is in a safe state since sequence <Pc, Pb, Pa> satisfies safety requirement.

Q5. Table below describe the snapshot of the system taken at particular time. Using banker's algorithm, determine if the system is in a safe state. (Note: Assume that the available resources for A, B and C are 4, 3, 2 respectively)

Processes	**Allocation**			**Maximum**		
	A	**B**	**C**	**A**	**B**	**C**
P1	0	1	1	6	6	1
P2	3	1	1	4	2	2
P3	3	0	1	5	3	3
P4	0	0	1	1	1	4
P5	2	1	1	5	1	5
P6	2	2	2	3	4	2

Answer:

Processes	**Allocation**			**Maximum**			**Need**			**Available**		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
										4	3	2
P2	3	1	1	4	2	2	1	1	1	7	4	3
P3	3	0	1	5	3	3	2	3	2	10	4	4
P4	0	0	1	1	1	4	1	1	3	10	4	5
P5	2	1	1	5	1	5	3	0	4	12	5	6
P6	2	2	2	3	4	2	1	2	2	14	7	8
P1	0	1	1	6	6	1	6	5	0	14	8	9

- The system is in a safe state since the sequence < P2, P3, P4, P5, P6, P1 > satisfies safety criteria

Q6. In some operating systems, process scheduling may be performed at 3 different levels, namely short-term, medium-term, and long-term scheduling.
For short-term scheduling, some commonly used CPU scheduling policies are First-Come-First-Served (FCFS), Shortest-Remaining-Time-First (SRTF), Round-Robin and Preemptive Priority –based scheduling.

(a) Which of the policies cause starvation of processes? Explain your answer.
- **Shortest-Remaining-Time-First (SRTF)** - SRTF may lead to starvation of processes which CPU will require a long time to complete if short processes are continually added.

- **Preemptive Priority-based Scheduling** - This will lead to starvation of processes because the low priority process can keep waiting indefinitely without executing

- Starvation avoidance: **Process aging** involves temporarily increases the priority of a low-priority process that has not been run in a while to ensure that it gets a chance to get scheduled to run.

(b) Differentiate between deadlock and starvation. Which situation is more serious?
- **Deadlock** is a problem occurring when the resources needed by some jobs to finish execution are held by other jobs, which, in turn, are waiting for other resources to become available.
- **Starvation** is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time
- Deadlock is more serious than starvation because it affects more than one job.

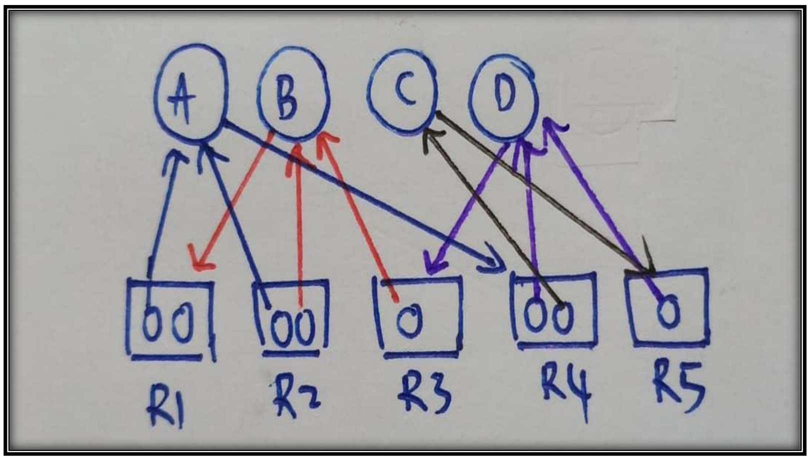
- **Deadlock** is set of blocked processes which each of them holding a resource and keeps waiting for another resource that held by another process.
- **Starvation** is a set of processes are waiting too long in the queue for execution.
- Deadlock is more serious than starvation.

Self-Review

Q1. There are four processes (A, B, C, and D) and five resources (R1, R2, R3, R4 and R5). Assume that all the resources are non-sharable and the numbers of instances for each resource are 2, 2, 1, 2, 1 respectively.

Process A holds one instance of R1 and R2. It requests one instance of R4.
Process B holds one instance of R2 and R3. It requests one instance of R1.
Process C holds one instance of R4. It requests another one instance of R5.
Process D holds one instance of R4 and R5. It requests one instance of R3.

(i) Draw a directed resources allocation graph for the above scenario.



(ii) Are these four processes in a deadlock state? Justify your answer.
- No deadlock.
- Process B can finish its process first. Then, it releases the instances of R3 to Process D. When Process D is completed. Process C can finish the process after getting the instance of R5. Finally, Process A can complete its process.

Q2. Assuming that there are 5 processes (P1, P2, P3, P4 and P5) which are processed by a CPU. While processing, the processes need 3 resources (A, B, C). The **Table 1** below has shown the resources allocated to each current process, maximum resources the processes require to complete their tasks and the available resources respectively. Use ***banker's algorithm*** to determine whether the system is in a safe state or unsafe state. Justify your answers.

	A	*B*	*C*
P1	*0*	*1*	*2*
P2	*2*	*0*	*0*
P3	*3*	*2*	*1*
P4	*1*	*2*	*3*
P5	*0*	*4*	*2*
Allocated resources			

	A	*B*	*C*
P1	*0*	*3*	*4*
P2	*3*	*3*	*4*
P3	*5*	*6*	*2*
P4	*2*	*3*	*4*
P5	*1*	*6*	*6*
Max resources required			

A	*B*	*C*
2	*2*	*2*
Available resources		

Table 1

Processes	**Allocation**			**Maximum**			**Need**			**Available**		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
										2	2	2
P1	0	1	2	0	3	4	0	2	2	2	3	4
P2	2	0	0	3	3	4	1	3	4	4	3	4
P4	1	2	3	2	3	4	1	1	1	5	5	7
P5	0	4	2	1	6	6	1	2	4	5	9	9
P3	3	2	1	5	6	2	2	4	1	8	11	10

- The system is in a safe state since the sequence < P1, P2, P4, P5, P3> satisfies safety criteria

Notes:

1. Available = available + allocation
2. Need = Maximum allocation – allocation
3. Total number of resources = allocation + available
4. The system is **not in the deadlock** as long as one process can execute even all process holding and waiting for some resources
   - Process can execute if the resource has free instance
   - Once the first process done, it will release holding resources
   - Then, the remaining process can be executed one by one
5. For the banker algorithms, if the available resources are not enough for its need, then it will check in ascending order until whole process is finish executing