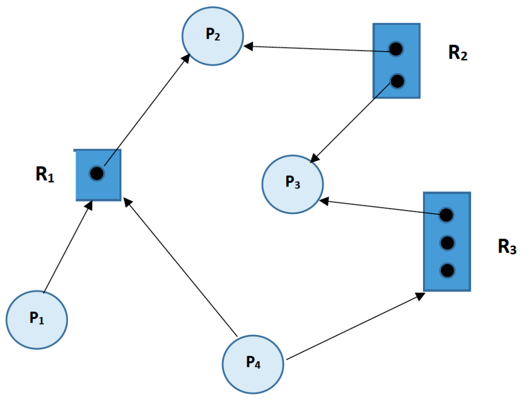
Tutorial 5 – Deadlocks (Chapter 8)

Operating Systems Comp Sci 3SH3, Winter 2024
Prof. Neerja Mhaskar

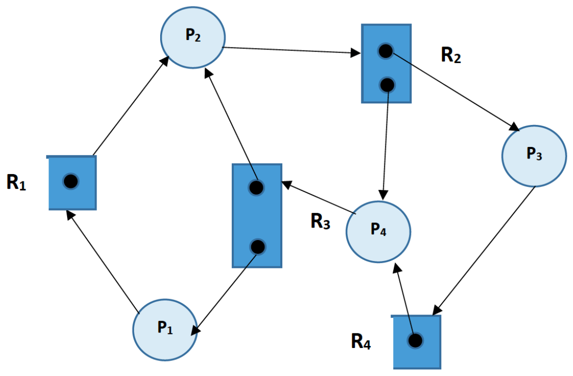
Questions:

1. Consider the below resource allocation graph. Is the system in a deadlock state? If so, report the cycle(s) causing deadlock. If not, explain the order in which processes access the resources requested and complete execution.



No deadlock. They complete execution in P2 -> P3 -> P1 -> P4

2. Consider the below resource allocation graph. Is the system in a deadlocked state? If so, report the cycle(s) causing deadlock. If not, explain the order in which processes access the resources requested and complete execution.



It is in a deadlock state. There is a cycles. It is
P1 -> P2 -> P3 -> P4 -> P1
P2 -> P3 -> P4 -> P2

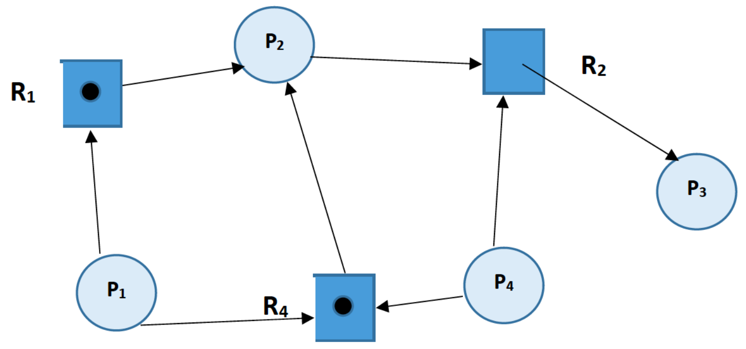
3. Consider the following snapshot of a system:

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

Answer the following questions using the banker's algorithm:

a. What is the content of the matrix Need? **Answered above**
b. Is the system in a safe state? **Yes**
c. If a request from process P_1 arrives for (0,4,2,0), can the request be granted immediately? **Yes, because it's less than the work matrix and less than the max need the process requires**

4. Consider the below resource allocation graph. Construct the corresponding wait-for graph. Is the system in deadlock? If so, provide the cycle causing deadlock.



$$\begin{tikzcd}
\textbf{P1} \arrow[r] & \textbf{P2} \arrow[d] \\
\textbf{P4} \arrow[ur] \arrow[r] & \textbf{P3}
\end{tikzcd}$$

5. Consider the following snapshot of a system at time *T*0:

Five processes ***P*₀** through ***P*₄**.

Three resource types A (10 instances), *B* (3 instances), and *C* (6 instances)

Snapshot at time ***T*₀**:

Finish = [false, false, false ,false, false] since there is no Allocation_i with (0 0 0).
Finish (after fulfillment) = [true, true, true, true, true]

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P0	2 1 1	0 0 0	0 0 0
P1	2 1 2	2 0 2	
P2	4 0 0	0 0 1	
P3	2 1 1	1 0 0	
P4	0 0 2	0 0 2	

P0
| Request (0 0 0) <= Available (0 0 0)
| Available = (2 1 1)
P2
| Request (0 0 1) <= Available (2 1 1)
| Available = (6 1 1)
P3
| Request (1 0 0) <= Available (6 1 1)
| Available = (8 2 2)
P4
| Request (0 0 2) <= Available (8 2 2)
| Available = (8 2 4)
P1
| Request (2 0 2) <= Available (8 2 4)
| Available = (10 3 6)

We'll use the deadlock detection algorithm for this since we're not given a max vector.

a) Is the system in deadlocked state? If no, provide a sequence of processes satisfying the safety requirement. If yes, explain why and list the processes involved in the deadlock. **No deadlock. P0 -> P2 -> P3 -> P4 -> P1**

b) Suppose process P1 makes an additional request of resource type B, the Request matrix is modified as follows:

	Request
	A B C
P_0	0 0 0
P_1	2 1 2
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

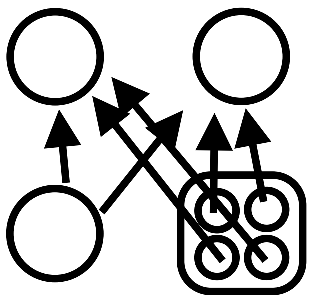
No deadlock since (2 1 2) <= (8 2 4)

c) Is the system in deadlocked state? If no, provide a sequence of processes satisfying the safety requirement. If yes, explain why and list the processes involved in the deadlock.

6. Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Show that the system is deadlock-free.

7. Consider the version of the dining-philosophers problem in which the chopsticks are placed at the center of the table and any two of them can be used by a philosopher. Assume that requests for chopsticks are made one at a time. Describe a simple rule for determining whether a particular request can be satisfied without causing deadlock given the current allocation of chopsticks to philosophers.

6th question:-



Case 1:
Two out of the three processes have acquired one resource each and we're left with only two more instances of the resource. In such case we would not have a deadlock since a single process remains and two resources remains and it can choose either of the remaining resource

Case 2:
One of the processes has acquired two instances, in such case we have two instances left and the remaining processes can acquire them which wouldn't cause a deadlock.

Case 3:
Two processes have acquired two instances each, which leaves us with no instance for the third process. In such case the process can acquire resources from either of the processes which finishes their execution first. Since the processes which had acquired resources weren't waiting for any of the other processes, they can finish execution without getting stuck in a deadlock.

Therefore the system in not in a deadlock

1.At most only 4 philosophers are allowed to sit

2. Asymmetric solution: Rewrite the code such that odd numbered philosopher picks left chopstick first whereas even numbered philosopher picks right chopstick first

3. A philosopher is only allowed to pick up a chopstick if both chopsticks besides him are available. In the case that even one of the chopsticks (either right or left) is not available, he can't pick up any of them. We can simulate this behaviour by blocking the part of the code where we pick up chopsticks in a mutex lock. This ensures whenever some other philosophers are trying to acquire the chopsticks, the other philosophers can't access it at the same time.

7th question:
Choose any one out of the three