COMP 346 – FALL 2019

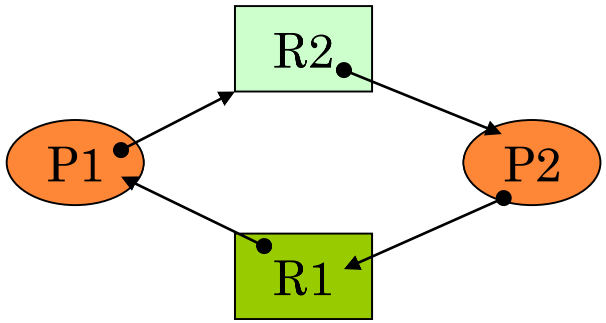
DEADLOCK

TOPICS

- Deadlock brief
  - Simplest Example
  - Necessary Deadlock Conditions
  - Resource Allocation Graph
  - Deadlock Handling
- Deadlock with Semaphores
- Examples

Deadlock

- Simplest example:
  - Two processes require two resources to complete (and release the resources).
  - There are only two instances of these resources.
  - If they acquire one resource each, they block indefinitely waiting for each other to release the other resource =>
    - **DEADLOCK**, one of the biggest problems in multiprogramming.



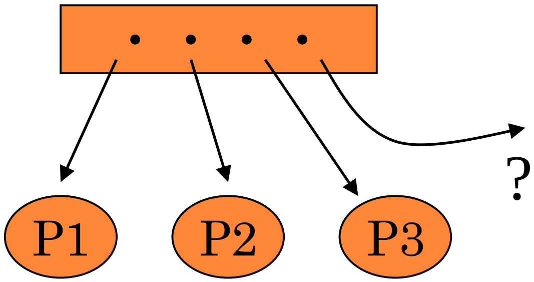
Necessary Deadlock Conditions

- Recall which conditions must hold :
  - **Mutual Exclusion**: at least one process exclusively uses a resource.
  - **Hold and wait**: a process possesses at least one resources and requires more, which are held by others.
  - **No preemption**: resources are released only in voluntary manner by processes holding them.
  - **Circular wait**: $P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow \ldots \rightarrow P_N \rightarrow P_1$.

Université Concordia University

Resource-Allocation Graph

- An easy way to illustrate resource allocation and visually detect the deadlock situation(s)
- Example:
  - N+1 resources
  - N processes
  - Every process needs 2 resources
  - Upon acquiring 2 resources, a process releases them
  - Is there a deadlock?



Handling Deadlocks

- Deadlock Ignorance
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection and Recovery

Université Concordia University

Deadlock Ignorance

- System designers pretend that deadlocks don’t happen.
- If they do happen, they don’t happen very often.
- Most common approach because it’s cheap (in terms of human labor, complexity of the system and system’s performance).
- Highest resource utilization.
- System administrator can simply kill deadlocked processes or restart the system if it’s not responding.

Deadlock Prevention

- Recall necessarily deadlock conditions.
- Deadlock prevention algorithms **assure at least one of these conditions do not hold**, thus *preventing* occurrence of the deadlock.
- Possible Disadvantages:
  - Low device utilization
  - Reduced system's throughput

Deadlock Avoidance

- Unlike deadlock prevention, deadlock avoidance algorithms do not watch for necessary deadlock conditions, but **use additional priority information about future resource requests**.
- This information will help the system to avoid entering the ***unsafe state***.
- Disadvantages: again, lower resource utilization (because resources may **not be granted** sometimes even if they are unused).

Université Concordia University

Deadlock Detection and Recovery

- Instead of preventing or avoiding deadlocks we **allow them to happen** and also provide **mechanisms to recover**.
- Problems:
  - How **often** do we run detection algorithms?
  - How **do** we recover?
  - What is the **cost** of detection and recovery?

Université Concordia University

1- Deadlock with Semaphores

- Semaphore alpha = new Semaphore(1);
- Semaphore beta = new Semaphore(1);

Thread 1:

```
alpha.Wait();
beta.Wait();

DoSomething();

beta.Signal();
alpha.Signal();
```

Thread 2:

```
beta.Wait();
alpha.Wait();

DoSomething();

alpha.Signal();
beta.Signal();
```

A Solution

```
Semaphore alpha = new Semaphore(1);
Semaphore beta  = new Semaphore(1);
```

Thread 1:

```
alpha.Wait();
beta.Wait();

DoSomething();

beta.Signal();
alpha.Signal();
```

Thread 2:

```
alpha.Wait();
beta.Wait();

DoSomething();

beta.Signal();
alpha.Signal();
```

UNIVERSITÉ Concordia UNIVERSITY

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	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	

UNIVERSITÉ Concordia UNIVERSITY

Example (Cont.)

The content of the matrix *Need* is defined to be *Max* – *Allocation*.

Available= Available + Allocation

	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

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

Example: P_1 Request (1,0,2)

Check that Request ≤ Available (that is, $(1,0,2) \le (3,3,2) \Rightarrow$ true.

	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 1	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

Executing safety algorithm shows that sequence $< P_1, P_3, P_4, P_0, P_2 >$ satisfies safety requirement.

Can request for (3,3,0) by P_4 be granted?

Can request for (0,2,0) by P_0 be granted?

Problem 1

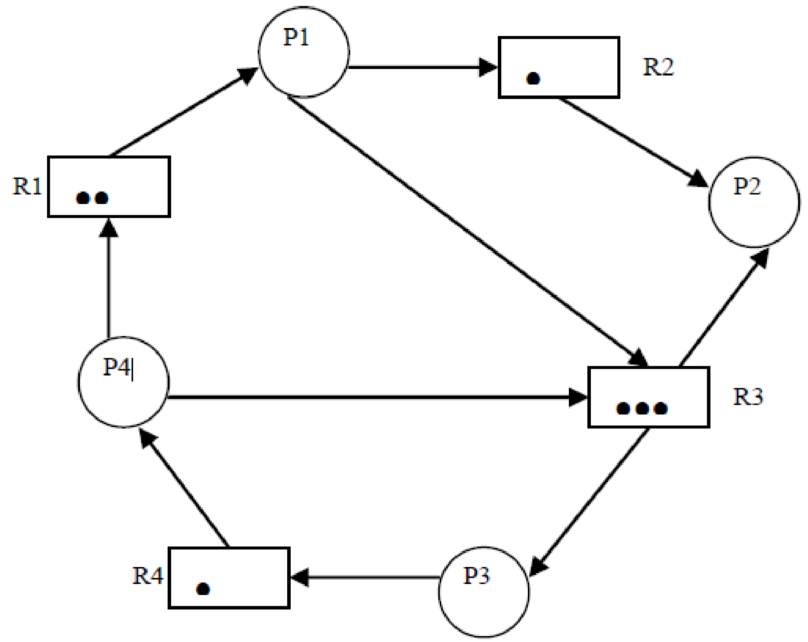
- In operating systems that implement multi-threading, the resources are allocated to the controlling process but it is the threads that can become deadlocked.
  - (i) What is the advantage to allocate the resources to the process?
  - (ii) What is the advantage that only the threads can be deadlocked?

Solution 1

- *(i) The resources can be shared by all the threads of a process.*
- *(ii) Blocking of a thread does not block the other threads.*

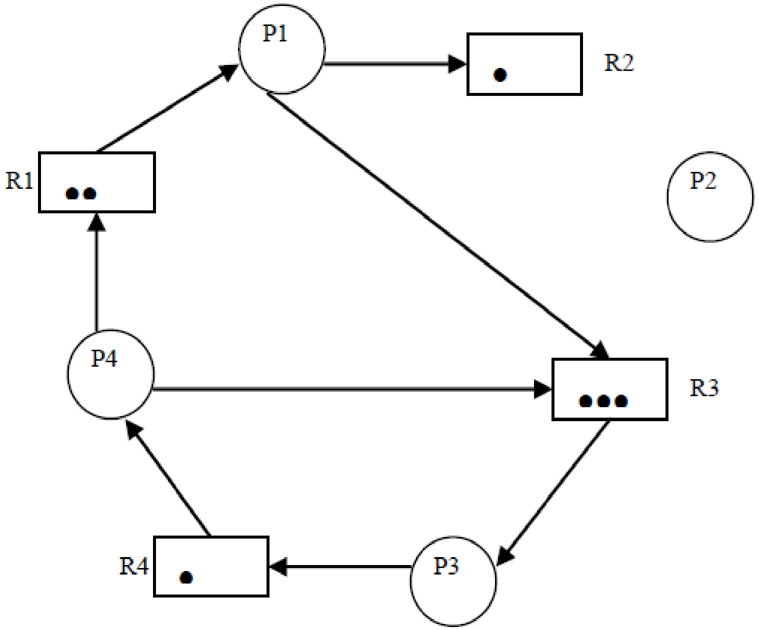
PROBLEM 2

- Consider the following resource allocation graph. Using the graph reduction method, show that there is no deadlock.

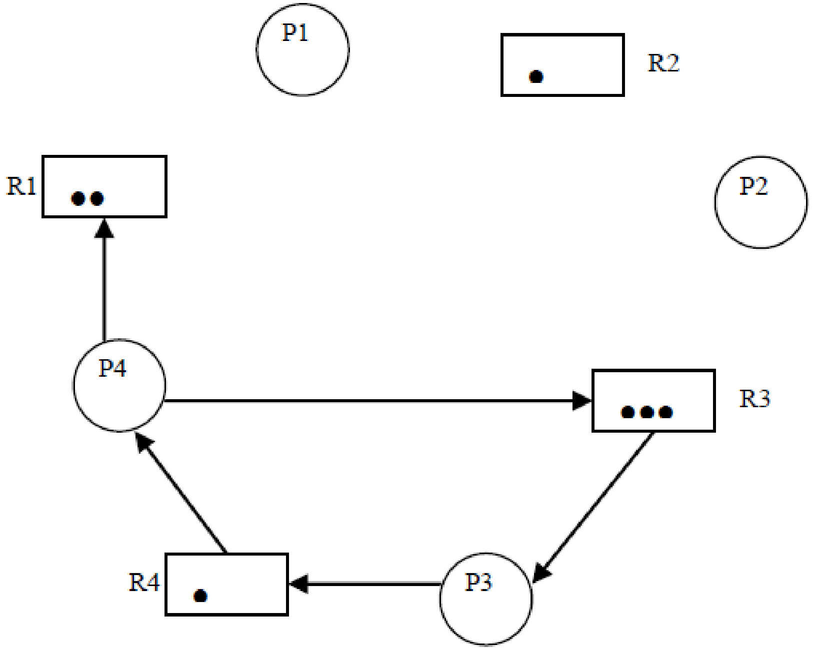


SOLUTION 2

- *Step 1 : reduce the graph by P2.*

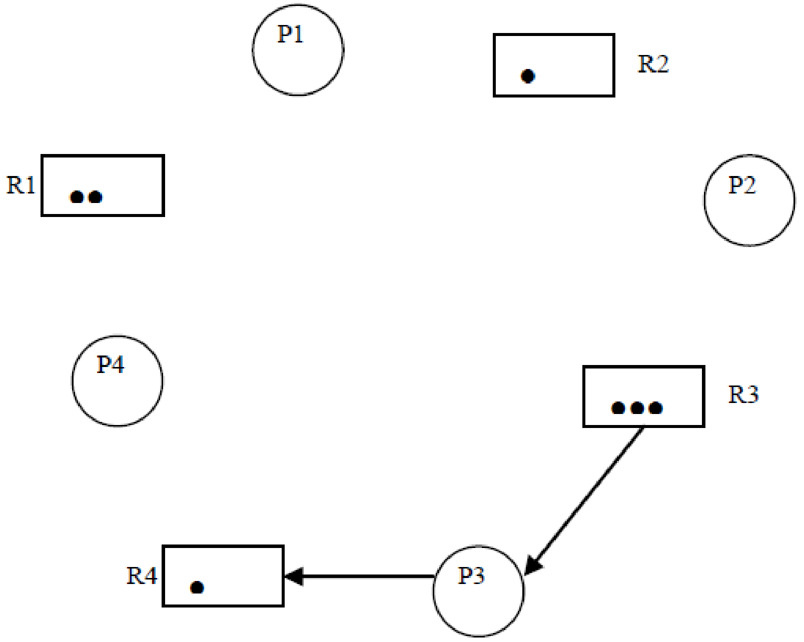


- *Step 2 : reduce the graph by P1.*

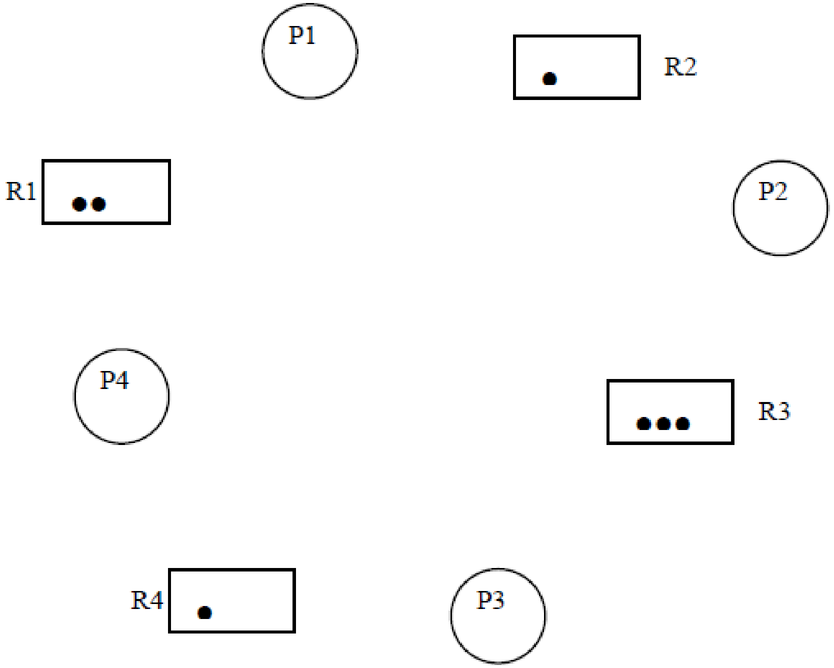


Solution 2 - Continue

- *Step 3 : reduce the graph by P4.*



- *Step 4 : reduce the graph by P3.*



PROBLEM 3

- Consider the following snapshot of a system:

	Allocation	Max	Available
	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
P_1	1 0 0 0	1 7 5 0	
P_2	1 3 5 4	2 3 5 6	
P_3	0 6 3 2	0 6 5 2	
P_4	0 0 1 4	0 6 5 6	

- Answer the following questions using the banker's algorithm:
- a. What is the content of the matrix **Need**?
- b. Is the system in a safe state?
- c. If a request from process *P*1 arrives for (0,4,2,0), can the request be granted immediately?

UNIVERSITÉ Concordia UNIVERSITY

SOLUTION 3

- a. The values of **Need** for processes $P0$ through $P4$ respectively are (0,0, 0, 0), (0, 7, 5, 0), (1, 0, 0, 2), (0, 0, 2, 0), and (0, 6, 4, 2).
- b. The system is in a safe state? Yes. With **Available** being equal to (1, 5, 2, 0), either process $P0$ or $P3$ could run. Once process $P3$ runs, it releases its resources, which allow all other existing processes to run.
- c. The request can be granted immediately? This results in the value of **Available** being (1, 1, 0, 0). One ordering of processes that can finish is $P0$, $P2$, $P3$, $P1$, and $P4$.