Chapter 7: Deadlocks

2020.6
Howon Kim

- 정보보호 및 지능형 IoT연구실 – http://infosec.pusan.ac.kr
- 부산대 지능형융합보안대학원 – http://aisec.pusan.ac.kr

Chapter 7: Deadlocks

- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance

Chapter Objectives

- To develop a description of deadlocks, which prevent sets of concurrent processes from completing their tasks
- To present a number of different methods for preventing or avoiding deadlocks in a computer system

System Model

- System consists of resources
- Resource types $R_1, R_2, \ldots, R_m$

  CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
  - **request**
  - **use**
  - **release**

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion**: only one process at a time can use a resource
- **Hold and wait**: a process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption**: a resource can be released only voluntarily by the process holding it, after that process has completed its task
- **Circular wait**: there exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

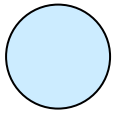
Resource-Allocation Graph(자원 할당 그래프)

A set of vertices V and a set of edges E.

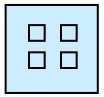
- **V is partitioned into two types**:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the **processes** in the system
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of **all resource types** in the system
- **request edge** (요청 edge) – directed edge $P_i \rightarrow R_j$
- **assignment edge** (할당 edge) – directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph (Cont.)

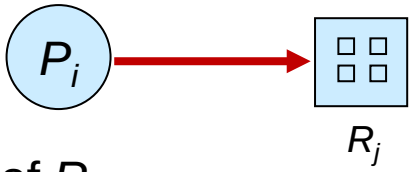
- Process



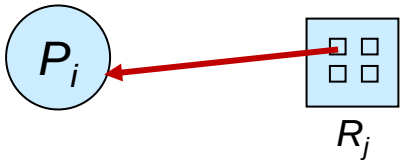
- Resource Type with 4 instances



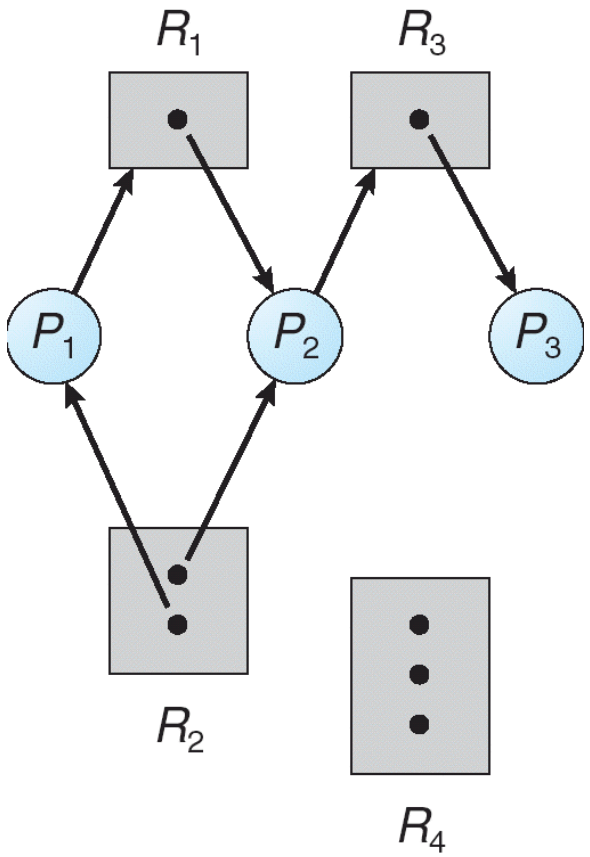
- P_i requests instance of R_j



- P_i is holding an instance of R_j

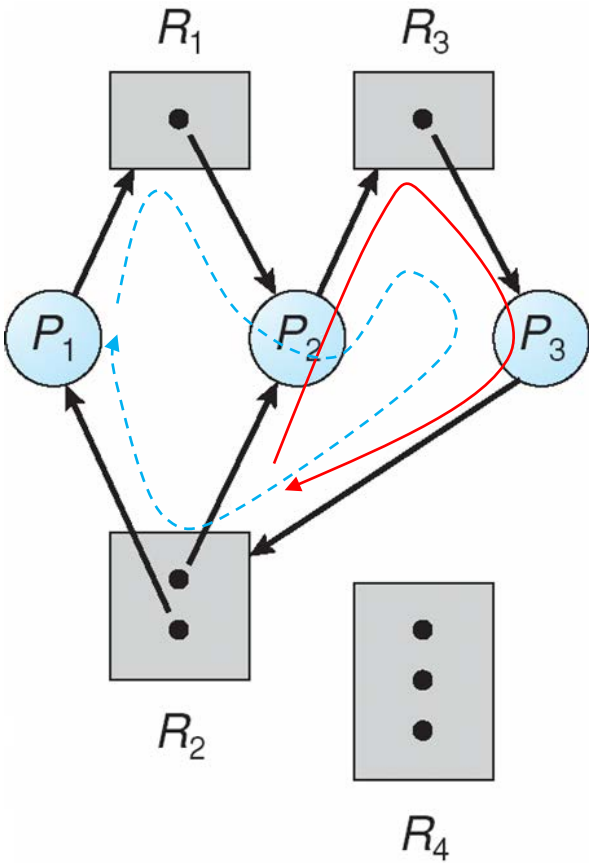


Example of a Resource Allocation Graph



- 프로세스 P1은 자원 R2의 한 인스턴스를 점유중이며, 자원 R1의 한 인스턴스를 요청하며 대기함
- 프로세스 P2는 R1과 R2의 한 인스턴스를 점유중이며, R3의 인스턴스 하나를 기다림
- 프로세스 P3는 R3의 인스턴스 한 개를 점유중

Resource Allocation Graph With A Deadlock



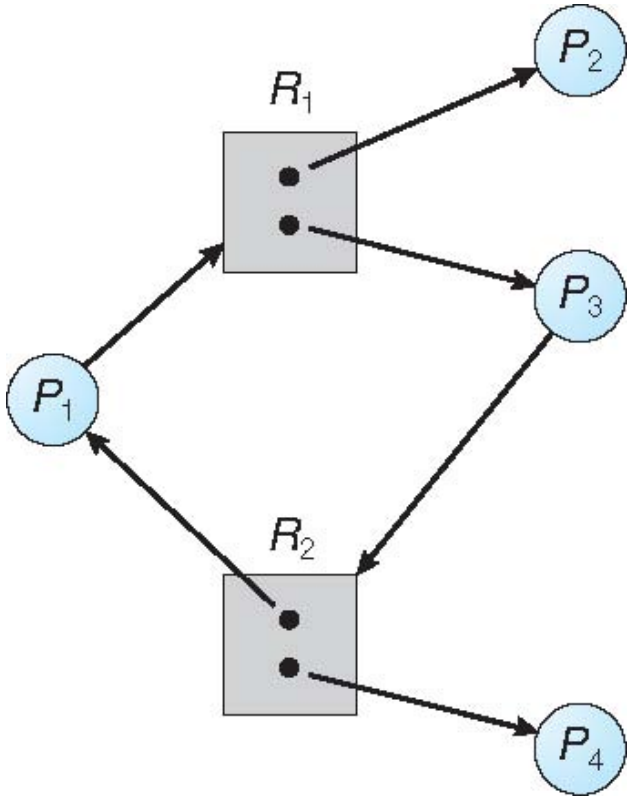
- 위 상황에서는 다음과 같은 두개의 사이클이 존재함

  P1 → R1 → P2 → R3 → P3 → R2 → P1

  P2 → R3 → P3 → R2 → P2

- P1, P2, P3는 교착상태임
- P2는 P3가 점유하는 자원 R3를 기다리고, P3는 P1&P2가 점유중인 자원 R2를 가다림. 또한, P1는 P2가 점유중인 R1을 기다림

Graph With A Cycle But No Deadlock



- 사이클이 존재하지만 Deadlock은 없음

  P1 → R1 → P3 → R2 → P1

- 사이클이 존재하지만 Deadlock은 발생하지 않음
- **P4가 점유중인 R2 인스턴스가 방출(release)될 수 있기 때문 → Release 되는 해당 R2의 인스턴스는 P3에 할당되어 사용될 수 있음 → 향후 cycle이 없어짐**

Basic Facts

- If graph contains no cycles $\Rightarrow$ no deadlock
- If graph contains a cycle $\Rightarrow$
  - if only one instance per resource type, then deadlock
  - if several instances per resource type, possibility of deadlock

Methods for Handling Deadlocks

- Three general approaches exist for dealing with deadlock:

Prevent Deadlock

- adopt a policy that eliminates one of the conditions

Avoid Deadlock

- make the appropriate dynamic choices based on the current state of resource allocation

Detect Deadlock

- attempt to detect the presence of deadlock and take action to recover

Deadlock Prevention Strategy

- Design a system in such a way that the possibility of deadlock is excluded
- Two main methods:
  - Indirect
    - prevent the occurrence of one of the three necessary conditions
  - Direct
    - prevent the occurrence of a circular wait

Deadlock Condition Prevention

- Mutual Exclusion을 없앨 수 있을까?
  - 시스템 설계시, mutual exclusion 을 사용하지 않을 수는 없음. (이는 shared resource의 일관성 유지에 필수적이므로)
- Hold and wait (점유대기)에 의한 Deadlock 발생 가능성 없애기
  - 프로세스가 자신이 사용할 모든 자원을 한꺼번에 요청하는 전략을 취하면 점유대기에 의한 Deadlock 발생을 없앨 수는 있음(단, 성능 저하 등 비효율적)
  - 즉,필요한 자원중에서 하나라도 할당받을 수 없는 **상황에서는 다른 자원을 Hold 하지 않음**
- No Preemption (비선점)에 의한 Deadlock 발생 가능성 없애기
  - 만약 자원을 점유한 프로세스가 다른 자원을 요청했을때, 할당 받을 수 없으면, 일단 자신이 점유한 자원을 반납함. 이후, 다시 예전에 점유했던 자원과 새로 원하는 자원을 다시 요청함
  - 한 프로세스에서 다른 프로세스가 점유한 자원을 원하면, 강제로 다른 프로세스가 점유한 자원을 강제로 반납시키고 이를 원하는 프로세스에 할당 **(선점 허용)**
- Circular Wait
  - define a linear ordering of resource types

Deadlock Avoidance

- Deadlock avoidance는 **(상호배제 조건, 점유대기조건, 비선점 조건)은 허용하고 자원을 할당할때, deadlock이 발생가능한 상황으로 진행하지 않도록 함**
  - Deadlock avoidance를 위해, process가 자원을 요청할 때, (현재 자원이 사용가능하다면), 있는 그대로 할당해주지 않고, 할당시 deadlock 발생 가능성이 있는지를 동적으로 체크함
    - Deadlock 발생 가능성 있으면 자원 할당해주지 않음
  - 이에, Deadlock avoidance를 위해선 자원 가용 개수와 프로세스의 자원 요구량 등을 미리 알고 있어야 함

- Deadlock avoidance를 위한 주요 두가지 방법
- 프로세스 자원 할당 거부**(Resource Allocation Denial)**
  - 수행 중인 프로세스가 요구하는 추가적인 자원 할당이 deadlock 발생 가능성 있으면, 자원 할당 하지 않음
- 프로세스 시작 거부 **(Process Initiation Denial)**
  - 프로세스가 시작하려 할때, 요구하는 자원 할당이 deadlock 발생 가능성 있으면, 프로세스를 시작하지 않음

Resource Allocation Denial(자원할당 거부)

- Referred to as the *banker's algorithm* (은행원 알고리즘)
- ***State*** of the system reflects the current allocation of resources to processes.
  - **시스템 상태:** 프로세스들이 자원을 요구하고 할당받은 관계를 의미함
- ***Safe state*** is one in which there is at least one sequence of resource allocations to processes that does not result in a deadlock.
  - **안전한 상태:** deadlock이 발생하지 않도록 프로세스에게 자원을 할당할 수 있는 진행경로가 존재함
- ***Unsafe state*** is a state that is not safe

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

Claim matrix **C**

요구 행렬

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

Allocation matrix **A**

할당 행렬

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

C – A

각 프로세스 Pi가 추가 요구하는 자원

R1	R2	R3
9	3	6

Resource vector **R**

자원 벡터

R1	R2	R3
0	1	1

Available vector **V**

가용 자원 벡터

(a) Initial state

Determination of a Safe State

Process i에 대하여, 다음을 만족시키면, 해당 프로세스는 완료 가능

$$C_{ij} - A_{ij} \leq V_j, \quad \text{for all } j$$

- P1은 R1, R2, R3를 각각 (2,2,2)개 더 요구하고 있음. 하지만, 현재 남은 자원은 (R1,R2,R3)=(0,1,1)이므로 P1은 수행 완료 불가
- P2는 (0,0,1) 요구하는데, 이는 제공 가능. 이때, P2가 완료되면 자신이 점유하던 자원 (6,1,2)을 모두 반납하여, 가용 자원은 (6,2,3)이 됨 **(See below)**

	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

각 프로세스 Pi가 추가 요구하는 자원

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
6	2	3

Available vector **V**

P2 실행후, Determination of a Safe 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**

각 프로세스 Pi가
추가 요구하는 자원

가용 자원이 (6,2,3) 있으므로, P1에게 할당하면, P1도 수행 완료 가능

	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**

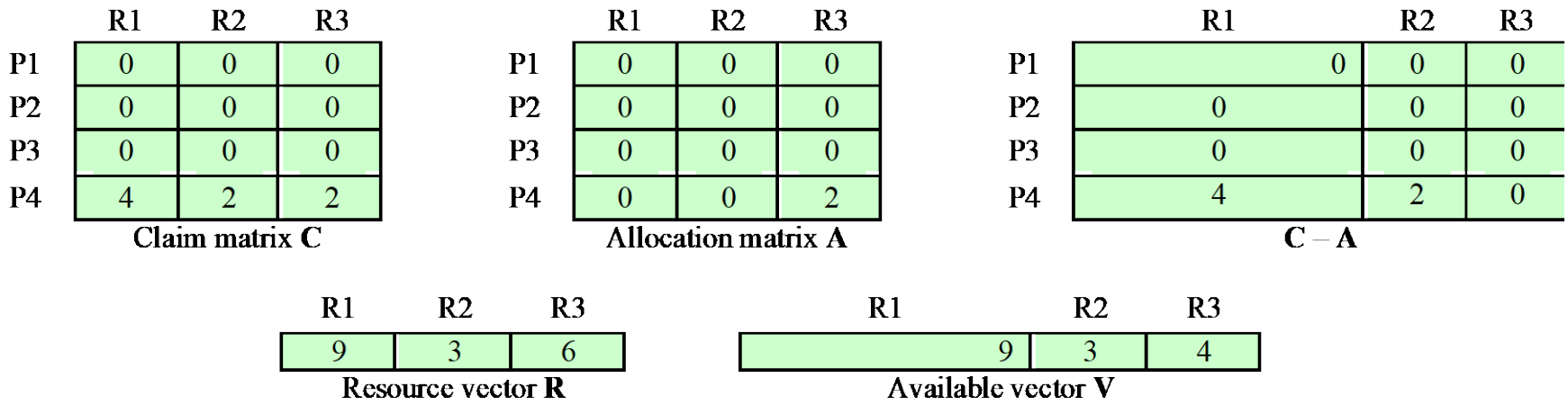
R1	R2	R3
7	2	3

Available vector **V**

(c) P1 runs to completion

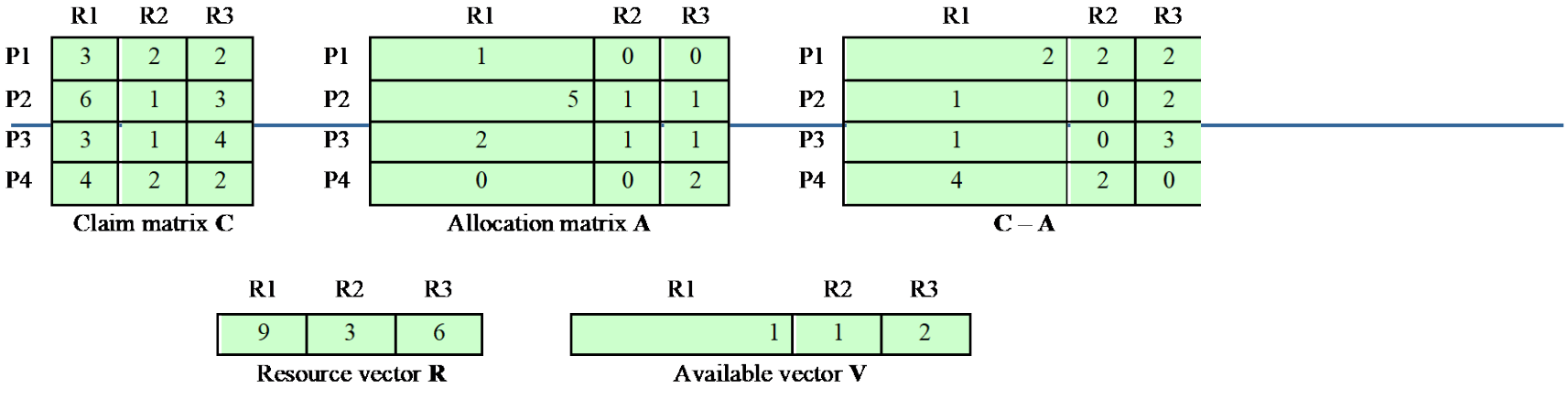
P1 실행후, Determination of a Safe State

- 이처럼 자원 할당시, safe state 이 계속 유지되면, deadlock 발생하지 않음
- Dijkstra가 제안한 “자원할당거부 방법”
- 즉, 프로세스가 자원 할당 요청시, 자원할당 결과가 시스템의 상태를 계속 안전 상태로 유지할 수 있는지 먼저 파악하여, 안전하면 할당 허용. 아니면 할당 거부!

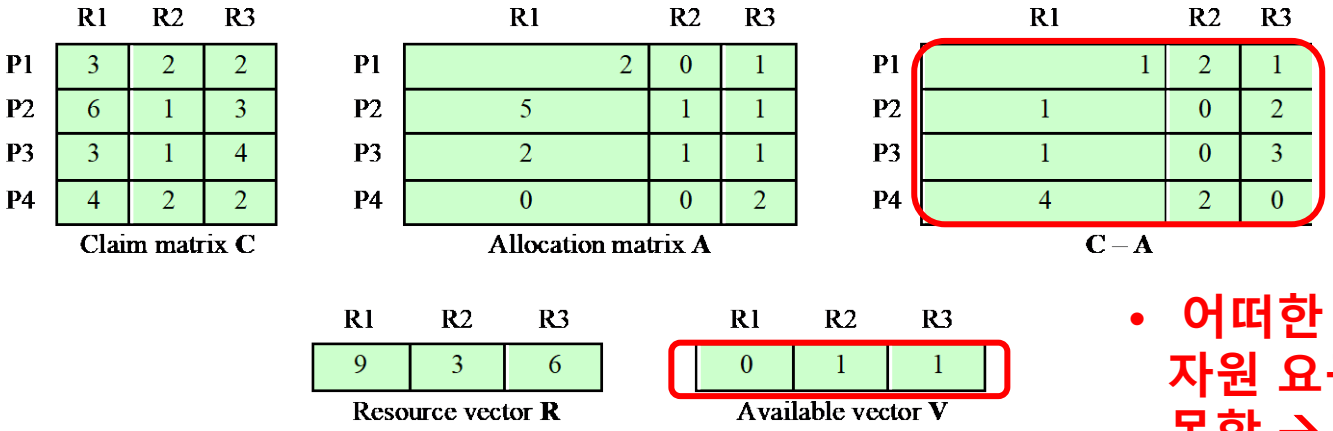


(d) P3 runs to completion

Figure 6.7 Determination of a Safe State



(a) Initial state



(b) P1 requests one unit each of R1 and R3

- 어떠한 process도 자신의 자원 요구를 모두 만족시키지 못함 → 수행을 완료하지 못하는 상태가 됨

Determination of an Unsafe State

Deadlock Avoidance Restrictions

- **Maximum resource requirement for each process** must be stated in advance
- Processes under consideration must be independent and with no synchronization requirements
- There must be a fixed number of resources to allocate
- No process may exit while holding resources

Deadlock Strategies

Deadlock prevention strategies are very conservative

- limit access to resources by imposing restrictions on processes

Deadlock detection strategies do the opposite

- resource requests are granted whenever possible