Chapter 06

교착 상태

Contents

학습목표

- 교착 상태의 정의와 발생 원인을 이해한다.
- 교착 상태가 발생하는 네 가지 필요조건을 알아본다.
- 교착 상태 해결 방법과 그 실효성을 알아본다.

1. 교착 상태의 개념

- 일상 생활에서 교착 상태

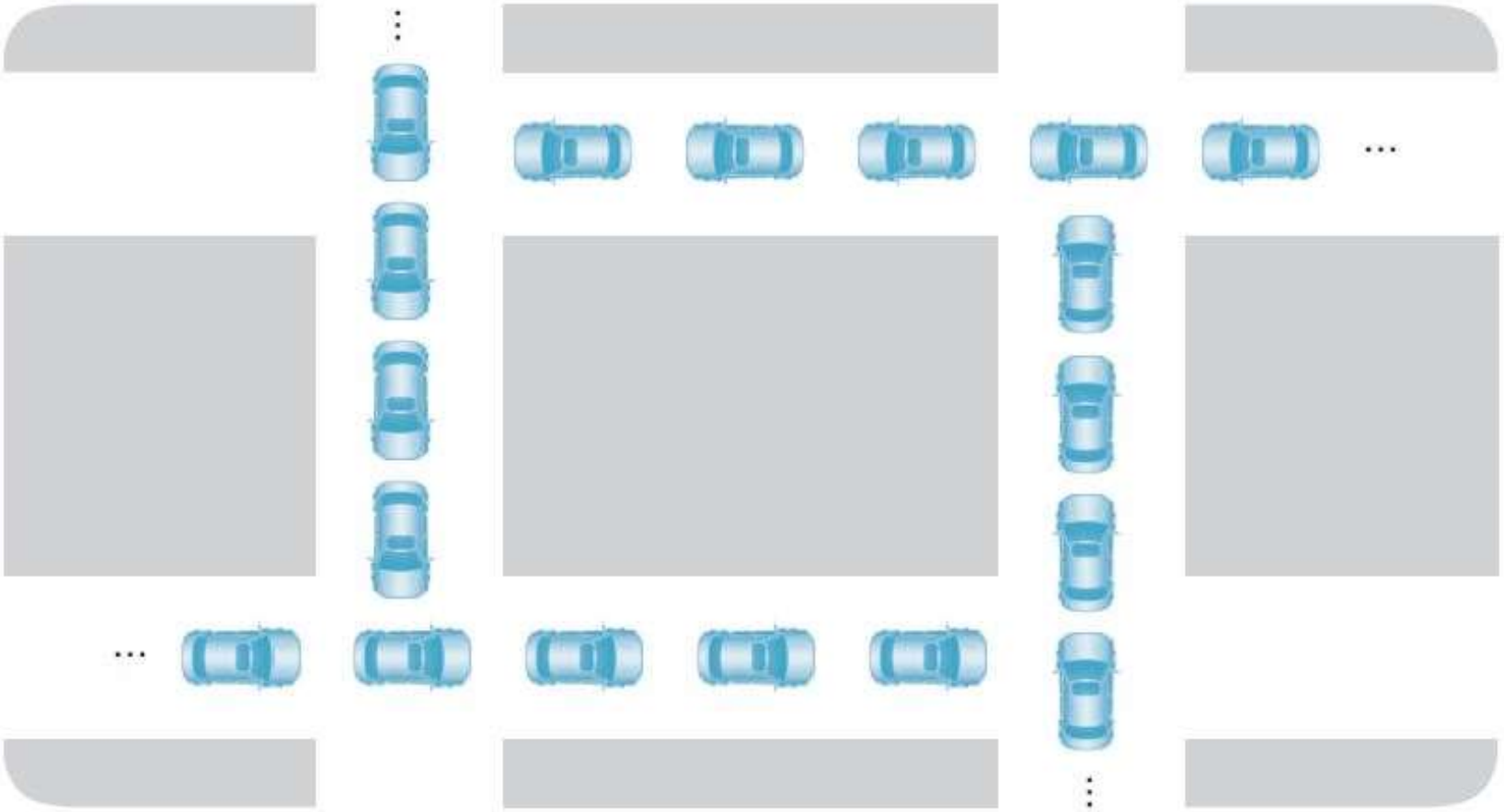


그림 5-2 교착 상태 예 : 교통마비 상태

교착 상태를 해결하기 위한 외부 간섭 필요

1 교착 상태의 정의

교착 상태 Deadlock

- 2개 이상의 프로세스가 다른 프로세스의 작업이 끝나기만 기다리며 작업을 더 이상 진행하지 못하는 상태

아사 상태Stravation와 차이점

- 아사 현상 : 운영체제가 잘못된 정책을 사용하여 특정 프로세스의 작업이 지연되는 문제
- 교착 상태 : 여러 프로세스가 작업을 동시에 진행하다 보니 자연적으로 일어나는 문제

그림 6-2 요리사의 자원 선점과 교착 상태

2 교착 상태의 발생

시스템 자원

- 교착 상태는 다른 프로세스와 공유할 수 없는 자원을 사용할 때 발생

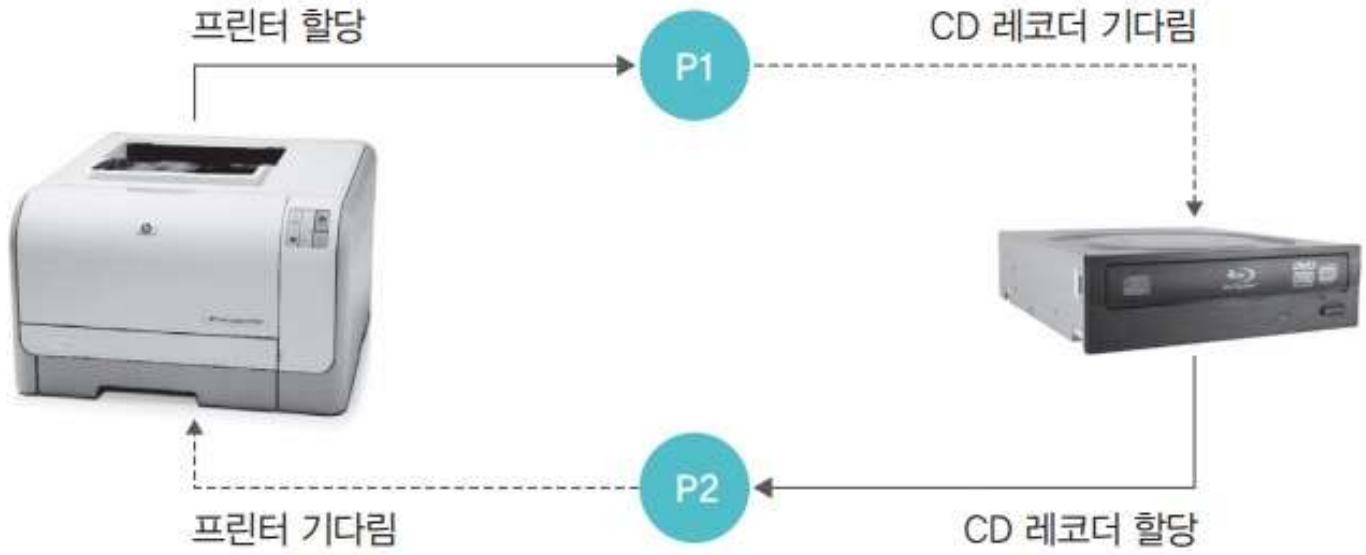


그림 6-3 시스템 자원 사용 도중 교착 상태 발생

2 교착 상태의 발생

공유 변수

- 교착 상태는 공유 변수를 사용할 때 발생

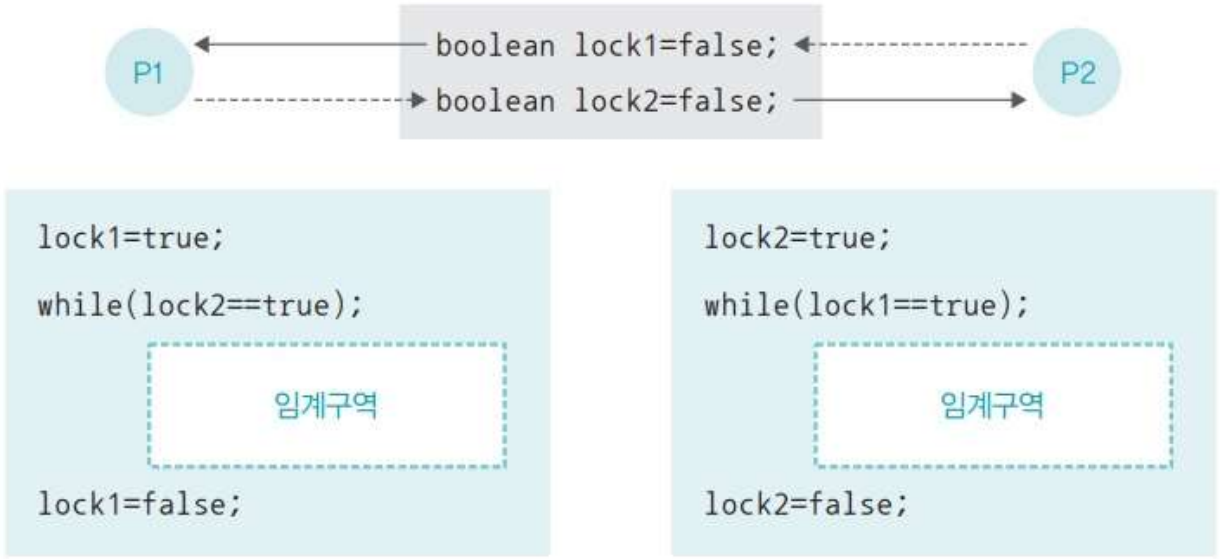


그림 6-4 교착 상태를 발생시키는 임계구역 코드

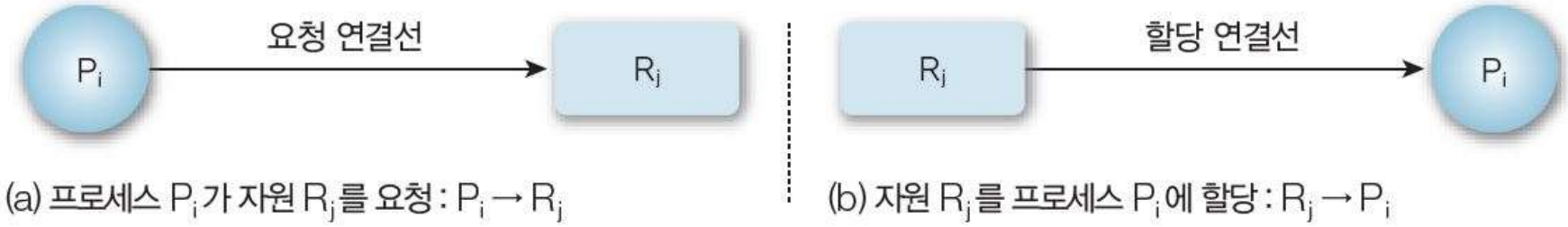
응용 프로그램

- 데이터베이스 같은 응용 프로그램에서도 교착 상태 발생
- 데이터베이스는 데이터의 일관성을 유지하기 위해 잠금을 사용하는데, 이때 교착 상태가 발생할 수 있음

3 자원 할당 그래프

자원 할당 그래프

- 프로세스가 어떤 자원을 사용 중이고 어떤 자원을 기다리고 있는지를 방향성이 있는 그래프로 표현한 것
- G= (V, E)로 구성, 정점 집합 V는 프로세스 집합 P= {P1, P2, ……, Pn}과 자원 집합 R= {R1, R2, ……, Rn}으로 나뉨. 간선 집합 E는 원소를 (Pi, Rj)나 (Rj, Pi)와 같은 순서쌍으로 나타냄
- **프로세스는 원으로, 자원은 사각형으로 표현**



(a) 프로세스 P_i가 자원 R_j를 요청 : $P_i \rightarrow R_j$

(b) 자원 R_j를 프로세스 P_i에 할당 : $R_j \rightarrow P_i$

그림 5-7 자원 할당 그래프

그림 6-7 요리사 문제의 자원 할당 그래프

3 자원 할당 그래프

다중 자원

- 여러 프로세스가 하나의 자원을 동시에 사용하는 경우
- 수용할 수 있는 프로세스 수를 사각형 안에 작은 동그라미로 표현

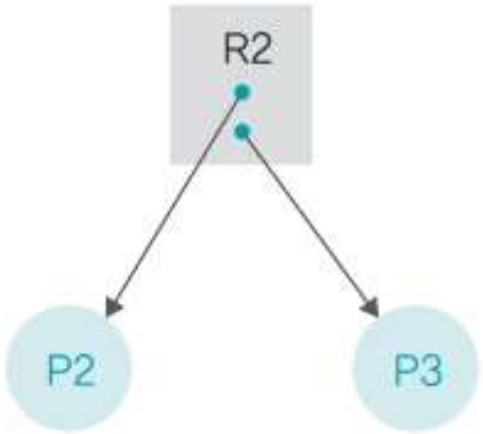


그림 6-6 2개의 프로세스를 수용할 수 있는 자원

3 자원 할당 그래프

식사하는 철학자 문제

- 왼쪽에 있는 포크를 잡은 뒤 오른쪽에 있는 포크를 잡아야만 식사 가능

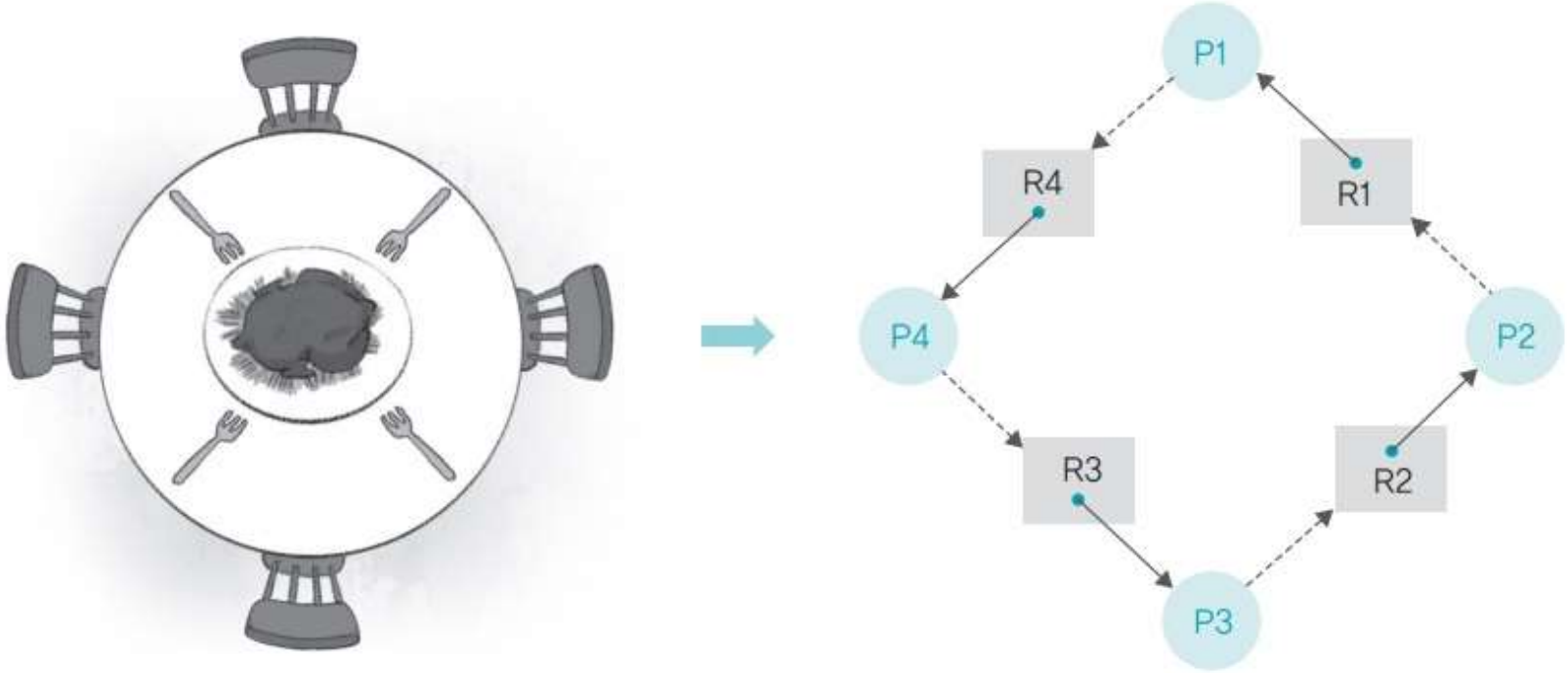


그림 6-8 식사하는 철학자 문제의 자원 할당 그래프

3 자원 할당 그래프

식사하는 철학자 문제에서 교착 상태가 발생하는 조건

❶ 철학자들은 서로 포크를 공유할 수 없음
→ 자원을 공유하지 못하면 교착 상태가 발생

❷ 각 철학자는 다른 철학자의 포크를 빼앗을 수 없음
→ 자원을 빼앗을 수 없으면 자원을 놓을 때까지 기다려야 하므로 교착 상태가 발생

❸ 각 철학자는 왼쪽 포크를 잡은 채 오른쪽 포크를 기다림
→ 자원 하나를 잡은 상태에서 다른 자원을 기다리면 교착 상태가 발생

❹ 자원 할당 그래프가 원형
→ 자원을 요구하는 방향이 원을 이루면 양보를 하지 않기 때문에 교착 상태가 발생

1 교착 상태 필요조건

교착 상태 필요(충분) 조건

- 상호 배제mutual exclusion : 한 프로세스가 사용하는 자원은 다른 프로세스와 공유할 수 없는 배타적인 자원이어야 함
- 비선점non-preemption : 한 프로세스가 사용 중인 자원은 중간에 다른 프로세스가 빼앗을 수 없는 비선점 자원이어야 함
- 점유와 대기hold and wait : 프로세스가 어떤 자원을 할당받은 상태에서 다른 자원을 기다리는 상태여야 함
- 원형 대기circular wait : 점유와 대기를 하는 프로세스 간의 관계가 원을 이루어야 함

1 교착 상태 필요조건

교착 상태 필요 조건 분석

- 상호 배제, 비선점 조건 : 자원이 어떤 특징을 가지는지를 나타냄
- 점유와 대기, 원형 대기 조건 : 프로세스가 어떤 행위를 하고 있는지를 나타냄

정의 6-1 교착 상태 필요조건

교착 상태는 상호 배제, 비선점, 점유와 대기, 원형 대기를 모두 충족해야 발생하고, 이 중 단 하나라도 충족하지 않으면 발생하지 않는다. 따라서 이 네 가지 조건을 교착 상태 필요조건이라고 한다.

2 식사하는 철학자 문제와 교착 상태 필요조건

식사하는 철학자 문제와 교착 상태 필요조건

- 상호 배제 : 포크는 한 사람이 사용하면 다른 사람이 사용할 수 없는 배타적인 자원임
- 비선점 : 철학자 중 어떤 사람의 힘이 월등하여 옆 사람의 포크를 빼앗을 수 없음
- 점유와 대기 : 한 철학자가 두 자원(왼쪽 포크와 오른쪽 포크)을 다 점유하거나, 반대로 두 자원을 다 기다릴 수 없음
- 원형 대기 : 철학자들은 둥그런 식탁에서 식사를 함, 원을 이룬다는 것은 선후 관계를 결정할 수 없어 문제가 계속 맴돈다는 의미(사각형 식탁에서 한 줄로 앉아서 식사를 한다면 교착 상태가 발생하지 않음)

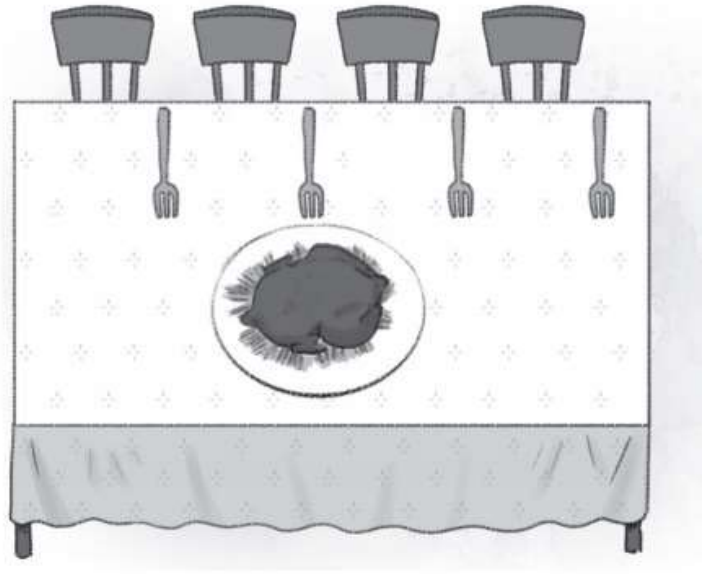
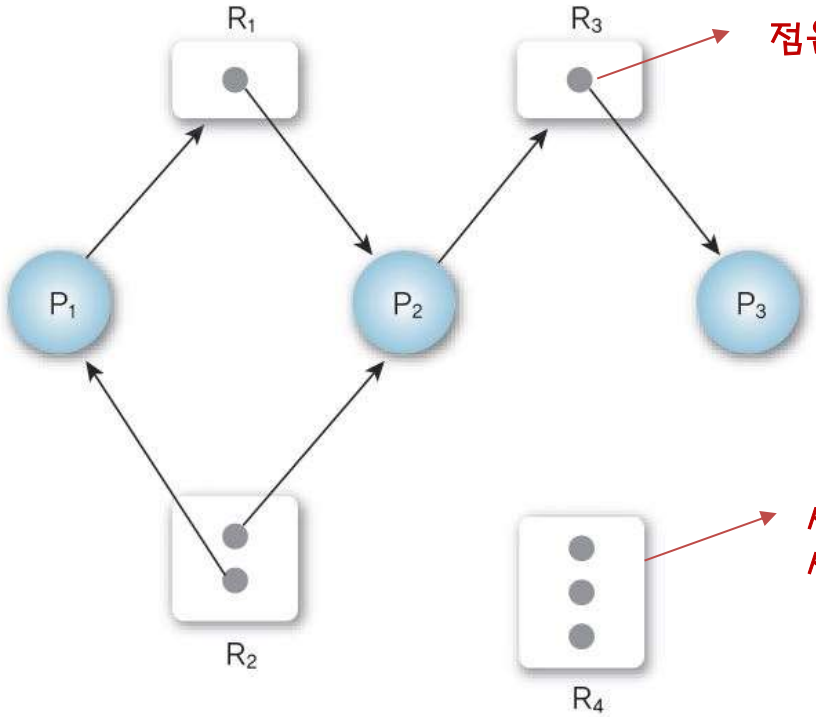


그림 6-10 사각형 식탁과 식사하는 철학자 문제

4. 교착 상태의 표현

- 자원 할당 그래프의 예



점은 자원 수를 뜻함

사각형 안의 점 : 할당 연결선
사각형 자체 : 요청 연결선

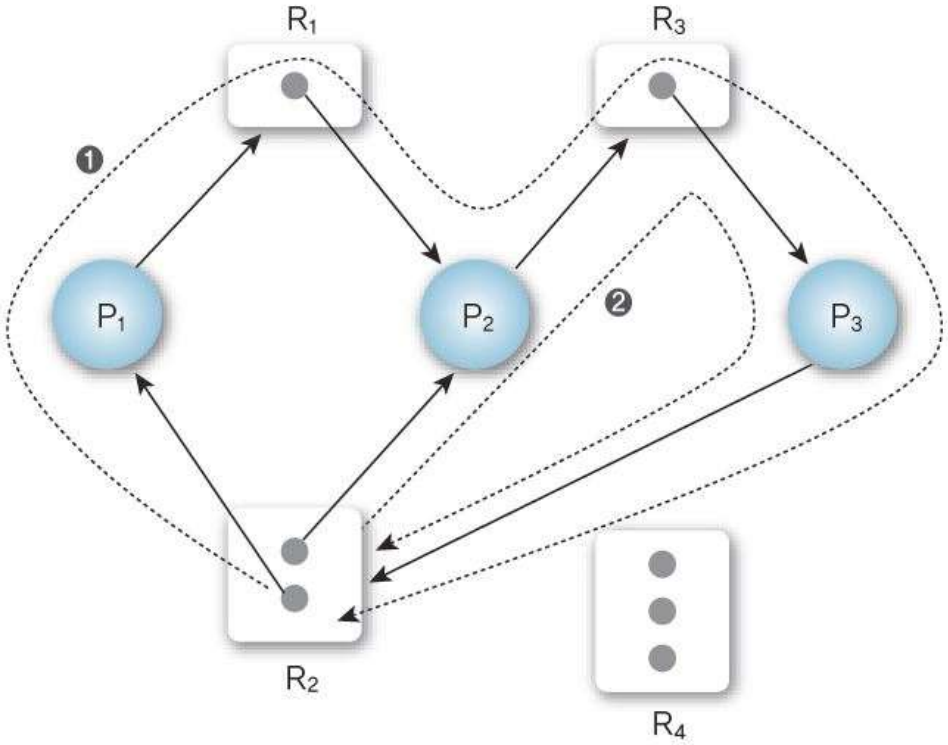
❶ 집합 P, R, E
- $P = \{P_1, P_2, P_3\}$
- $R = \{R_1, R_2, R_3, R_4\}$
- $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$

❷ 프로세스의 상태
- 프로세스 P_1은 자원 R_2의 자원을 하나 점유하고, 자원 R_1을 기다린다.
- 프로세스 P_2는 자원 R_1과 R_2의 자원을 각각 하나씩 점유하고, 자원 R_3을 기다린다.
- 프로세스 P_3은 자원 R_3의 자원 하나를 점유 중이다.

그림 5-9 자원 할당 그래프 예와 프로세스 상태

4. 교착 상태의 표현

- 교착 상태의 할당 그래프와 사이클



(a) 교착 상태의 할당 그래프

❶ $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

❷ $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

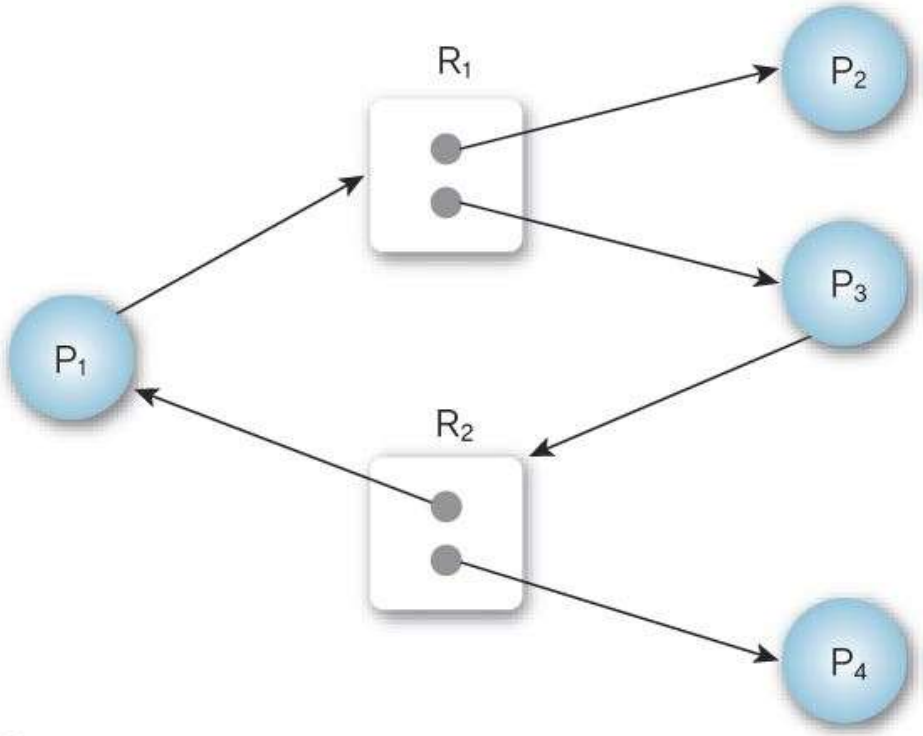
(b) (a)에 있는 사이클

그림 5-10 교착 상태의 할당 그래프와 사이클

그래프에 있는 사이클은 교착 상태 발생의 필요 조건이지 충분조건 아님

4. 교착 상태의 표현

- 사이클이 있으나 교착 상태가 아닌 할당 그래프



(a) 교착 상태의 할당 그래프

$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

(b) (a)에 있는 사이클

그림 5-11 사이클이 있으나 교착 상태가 아닌 할당 그래프

자원 할당 그래프에 사이클이 없다면 교착 상태 아님
그러나 사이클이 있다면 시스템은 교착 상태일 수도 있고 아닐 수도 있음

1 교착 상태 해결 방법

교착 상태 해결 방법

- 교착 상태 예방deadlock prevention : 교착 상태를 유발하는 네 가지 조건이 발생하지 않도록 무력화하는 방식
- 교착 상태 회피deadlock avoidance : 자원 할당량을 조절하여 교착 상태를 해결하는 방식
- 교착 상태 검출과 회복deadlock detection and recovery : 교착 상태 검출은 어떤 제약을 가하지 않고 자원 할당 그래프를 모니터링하면서 교착 상태가 발생하는지 살펴보는 방식으로 만약 교착 상태가 발생하면 교착 상태 회복 단계가 진행됨

2 교착 상태 예방

상호 배제 예방

- 시스템 내에 있는 상호 배타적인 모든 자원, 즉 독점적으로 사용할 수 있는 자원을 없애버리는 방법
- 현실적으로는 모든 자원을 공유할 수 없으며 상호 배제를 적용하여 보호해야 하는 자원이 있음
- 상호 배제를 무력화하는 것은 사실상 어려움

비선점 예방

- 모든 자원을 빼앗을 수 있도록 만드는 방법
- 그러나 아사 현상을 일으켜 비선점 조건을 무력화하기는 어려움

2 교착 상태 예방

점유와 대기 예방

- 프로세스가 자원을 점유한 상태에서 다른 자원을 기다리지 못하게 하는 방법
- '전부 할당하거나 아니면 아예 할당하지 않는' 방식을 적용
- 자원이 아닌 프로세스의 자원 사용 방식을 변화시켜 교착 상태를 처리한다는 점에서 의미가 있음

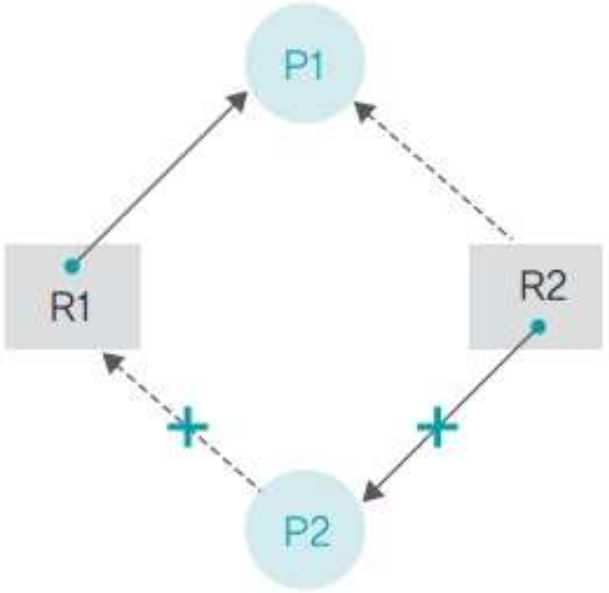


그림 6-11 전부 할당하거나 아니면 아예 할당하지 않는 방식

2 교착 상태 예방

점유와 대기 예방의 단점

- 프로세스가 자신이 사용하는 모든 자원을 자세히 알기 어려움
- 자원의 활용성이 떨어짐
- 많은 자원을 사용하는 프로세스가 적은 자원을 사용하는 프로세스보다 불리함
- 결국 일괄 작업 방식으로 동작

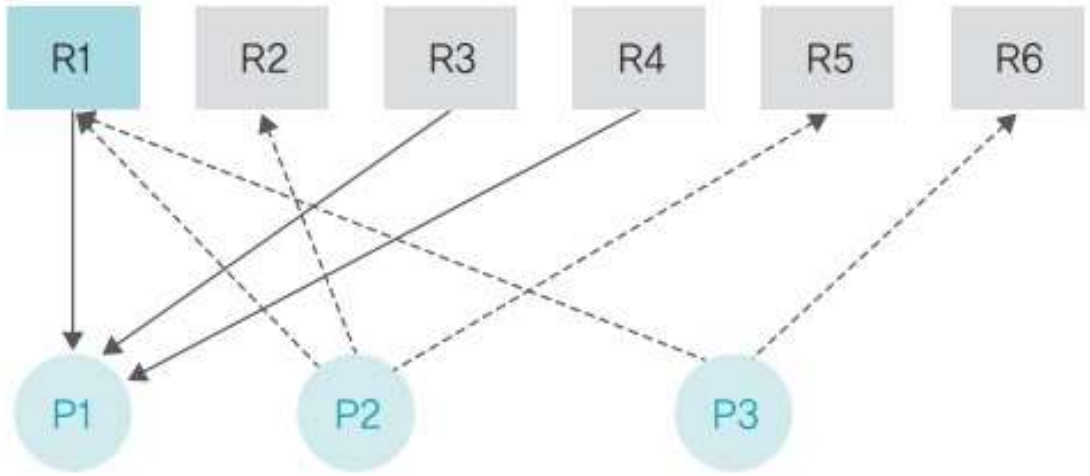


그림 6-12 필요한 자원을 모두 할당한 후 실행

2 교착 상태 예방

원형 대기 예방

- 점유와 대기를 하는 프로세스들이 원형을 이루지 못하도록 막는 방법
- 모든 자원에 숫자를 부여하고 숫자가 큰 방향으로만 자원을 할당하는 것

[예] 마우스를 할당받은 상태에서 프린터를 할당받을 수는 있지만 프린터를 할당받은 상태에서는 마우스나 하드디스크를 할당받을 수 없음

그림 6-13 자원에 대한 번호 매김

2 교착 상태 예방

원형 대기 예방과 교착 상태 해결

- 프로세스 P2는 자원을 할당받을 수 없어 강제 종료되고 프로세스 P1은 정상적으로 실행

원형 대기 예방의 단점

- 프로세스 작업 진행에 유연성이 떨어짐
- 자원의 번호를 어떻게 부여할 것인지가 문제

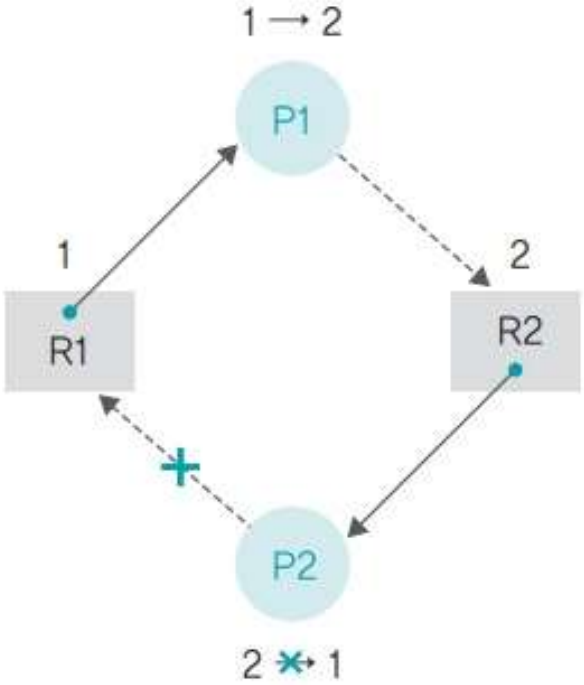


그림 6-14 원형 대기 조건의 부정

2 교착 상태 예방

교착 상태 예방 정리

- 교착 상태를 유발하는 네 가지 조건이 일어나지 않도록 제약을 가하는 방법
- 자원을 보호하기 위해 상호 배제와 비선점을 예방하기 어려움
- 점유와 대기, 원형 대기는 프로세스 작업 방식을 제한하고 자원을 낭비하기 때문에 사용할 수 없음

3 교착 상태 회피

교착 상태 회피의 개념

- 프로세스에 자원을 할당할 때 어느 수준 이상의 자원을 나누어 주면 교착 상태가 발생하는지 파악하여 그 수준 이하로 자원을 나누어 주는 방법
- 교착 상태가 발생하지 않는 범위 내에서만 자원을 할당하고, 교착 상태가 발생하는 범위에 있으면 프로세스를 대기시킴
- 즉, 할당되는 자원의 수를 조절하요 교착 상태를 피함

3 교착 상태 회피

안정 상태와 불안정 상태

- 교착 상태 회피는 자원의 총수와 현재 할당된 자원의 수를 기준으로 시스템을 안정 상태safe state와 불안정 상태unsafe state로 나누고 시스템이 안정 상태를 유지하도록 자원을 할당
- 할당된 자원이 적으면 안정 상태가 크고, 할당된 자원이 늘어날수록 불안정 상태가 커짐
- 교착 상태는 불안정 상태의 일부분이며, 불안정 상태가 커질수록 교착 상태가 발생할 가능성이 높아짐
- 교착 상태 회피는 안정 상태를 유지할 수 있는 범위 내에서 자원을 할당함 으로써 교착 상태를 피함

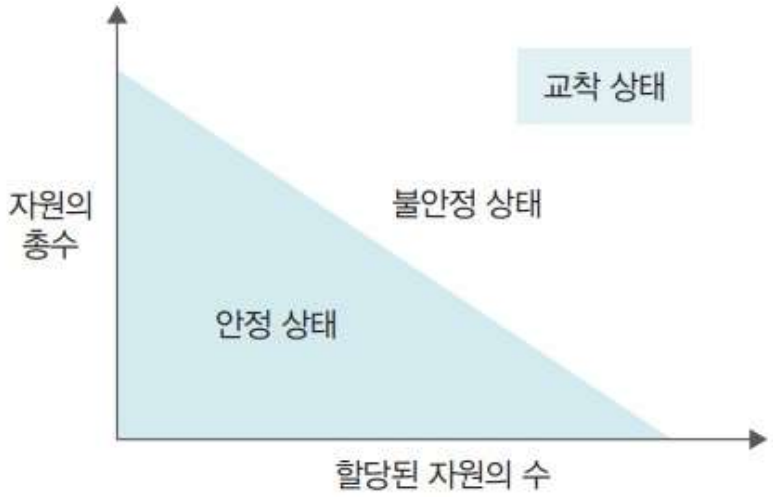


그림 6-15 안정 상태와 불안정 상태

3 교착 상태 회피

은행원 알고리즘

- 교착 상태 회피를 구현하는 대표적인 알고리즘
- 은행이 대출을 해주는 방식, 즉 대출 금액이 대출 가능한 범위 내이면(안정 상태이면) 허용되지만 그렇지 않으면 거부되는 것과 유사한 방식

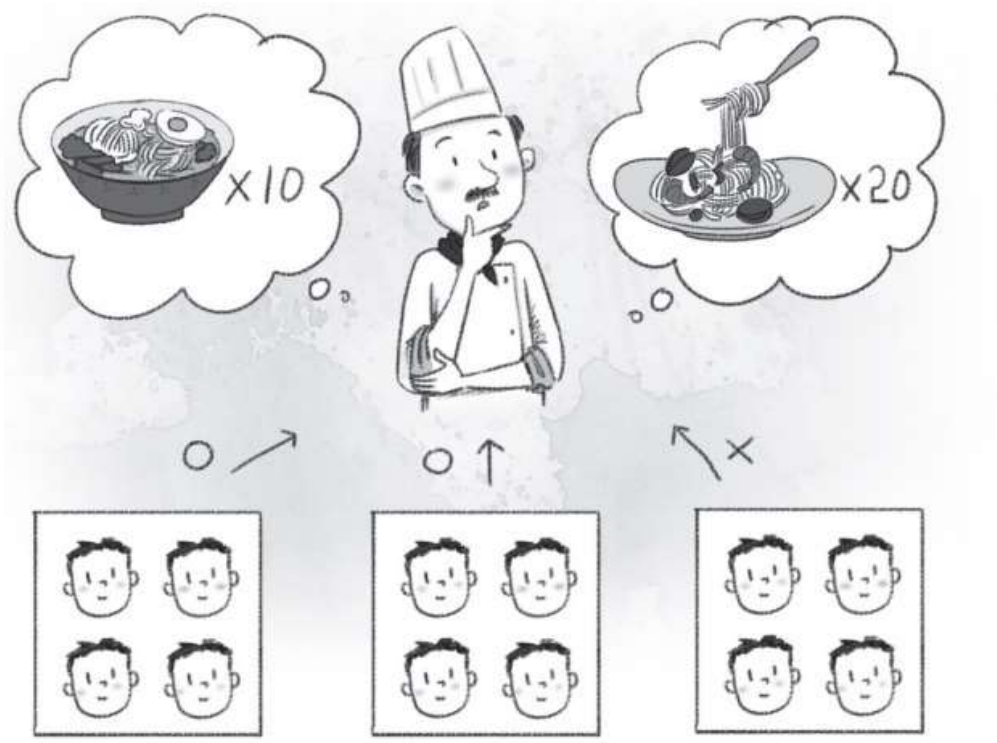


그림 6-16 은행원 알고리즘

3 교착 상태 회피

표 6-2 은행원 알고리즘의 변수

변수	설명
전체 자원(Total)	시스템 내 전체 자원의 수
가용 자원(Available)	시스템 내 현재 사용할 수 있는 자원의 수(가용 자원=전체 자원−모든 프로세스의 할당 자원)
최대 자원(Max)	각 프로세스가 선언한 최대 자원의 수
할당 자원(Allocation)	각 프로세스에 현재 할당된 자원의 수
기대 자원(Expect)	각 프로세스가 앞으로 사용할 자원의 수(기대 자원=최대 자원−할당 자원)

3 교착 상태 회피

은행원 알고리즘에서 자원 할당 기준

- 각 프로세스의 기대 자원과 비교하여 가용 자원이 하나라도 크거나 같으면 자원을 할당
- 가용 자원이 어떤 기대 자원보다 크지 않으면 할당하지 않음

정의 6-2 안정 상태

각 프로세스의 기대 자원과 비교하여 가용 자원이 크거나 같은 경우가 한 번 이상인 경우를 말한다.

3 교착 상태 회피

안정 상태의 예

Total=14 Available=2

Process	Max	Allocation	Expect
P1	5	2	3
P2	6	4	2
P3	10	6	4

그림 6-17 은행원 알고리즘(안정 상태)

불안정 상태의 예

Total=14 Available=1

Process	Max	Allocation	Expect
P1	7	3	4
P2	6	4	2
P3	10	6	4

그림 6-18 은행원 알고리즘(불안정 상태)

2. 교착 상태 회피

은행가 알고리즘 구현을 위한 자료구조

- **Available**
  - 각 형태별로 사용 가능한 자원 수(사용 가능량) 표시하는 길이가 m인 벡터
  - Available[j]=k이면, 자원을 k개 사용할 수 있다는 의미
- **Max**
  - 각 프로세스 자원의 최대 요청량(최대 요구량)을 표시하는 n×m 행렬
  - Max[i, j] =k이면, 프로세스 P는 자원이 R인 자원을 최대 k개까지 요청할 수 있다는 의미
- **Allocation**
  - 현재 각 프로세스에 할당되어 있는 각 형태의 자원 수(현재 할당량) 정의하는 n×m 행렬.
    Allocation[i, j]=k이면, 프로세스 P는 자원이 R인 자원을 최대 k개 할당받고 있다는 의미
- **Need**
  - 각 프로세스에 남아 있는 자원 요청(추가 요구량) 표시하는 n×m 행렬
  - Need[i, j] =k이면, 프로세스 P는 자신의 작업을 종료하려고 자원 R를 k개 더 요청한다는 의미

  Need[i, j] = Max[i, j] - Allocation[i, j] 라는 식 성립

- n : 시스템의 프로세스 수, m : 자원 수

2. 교착 상태 회피

■ 프로세스 P_i가 자원 요청 시 일어나는 동작

- 1단계 : $Request_i \le Need_i$이면 2단계로 이동하고, 그렇지 않으면 프로세스가 최대 요청치를 초과하기 때문에 오류 상태가 된다.
- 2단계 : $Request_i \le Available$이면 3단계로 이동하고, 그렇지 않으면 자원이 부족하기 때문에 P_i는 대기한다.
- 3단계 : 시스템은 상태를 다음과 같이 수정하여 요청된 자원을 프로세스 P_i에 할당한다.

$$Available = Available - Request_i;$$

$$Allocation_i = Allocation_i + Request_i;$$

$$Need_i = Need_i - Request_i;$$

그림 5-15 은행가 알고리즘

자원 할당이 안정 상태라면 처리가 되고 있는 프로세스 P_i는 자원 할당받음
불안정 상태라면 P_i는 $Request_i$를 대기하고 이전 자원 할당의 상태로 복귀

2. 교착 상태 회피

■ 안전 알고리즘의 시스템 상태 검사

- 1단계 : Work와 Finish를 각각 길이가 m과 n인 벡터라고 하자. Work = Available, Finish[i] = false, i = 1, 2, ……, n이 되도록 초기화한다.
- 2단계 : 다음 조건을 만족하는 i 값을 찾는다. i 값이 없으면 4단계로 이동한다.

  $Finish[i] == false$

  $Need_i \leq Work$
- 3단계 : 다음을 수행하고 2단계로 이동한다.

  $Work = Work + Allocation_i$

  $Finish[i] = true$
- 4단계 : 모든 i에 대하여 Finish[i] == true이면 시스템은 안정 상태이다.

그림 5-16 안전 알고리즘

2. 교착 상태 회피

- **시간 t_0 일때 시스템의 상태** 안정상태 시스템 자원의 수 = [8, 5, 9, 7]

프로세스	Allocation	Max	Need	Available
	ABCD	ABCD	ABCD	ABCD
P_0	2011	3214		1222
P_1	0121	0252		
P_2	4003	5105		
P_3	0210	1530		
P_4	1030	3033		
할당량	7375			

그림 5-17 시간 t_0일 때 시스템의 상태

- 1단계 : Work와 Finish를 각각 길이가 m과 n인 벡터라고 하자. Work = Available, Finish[i] = false, ……, n이 되도록 초기화한다.
- 2단계 : 다음 조건을 만족하는 i 값을 찾는다. i 값이 없으면 4단계로 이동한다.

  Finish[i] == false

  $Need_i \leq Work$
- 3단계 : 다음을 수행하고 2단계로 이동한다.

  $Work = Work + Allocation_i$

  Finish[i] = true
- 4단계 : 모든 i에 대하여 Finish[i] == true이면 시스템은 안정 상태이다.

2. 교착 상태 회피

- **시간 t_0 일 때 시스템의 새로운 상태** 시스템 자원의 수 = [8, 5, 9, 7]

프로세스	Allocation	Max	Need	Available
	ABCD	ABCD	ABCD	ABCD
P_0	2011	3214		1121
P_1	0222	0252		
P_2	4003	5105		
P_3	0210	1530		
P_4	1030	3033		
할당량	7476			

그림 5-17 시간 t_0일 때 시스템의 상태

- 1단계 : Work와 Finish를 각각 길이가 m과 n인 벡터라고 하자. Work = Available, Finish[i] = false, ……, n이 되도록 초기화한다.
- 2단계 : 다음 조건을 만족하는 i 값을 찾는다. i 값이 없으면 4단계로 이동한다.

  Finish[i] == false

  $Need_i \leq Work$
- 3단계 : 다음을 수행하고 2단계로 이동한다.

  $Work = Work + Allocation_i$

  Finish[i] = true
- 4단계 : 모든 i에 대하여 Finish[i] == true이면 시스템은 안정 상태이다.

3 교착 상태 회피

교착 상태 회피의 문제점

- 프로세스가 자신이 사용할 모든 자원을 미리 선언해야 함
- **교착 상태 회피 알고리즘을 실행하면 시스템 과부하 증가**
- 시스템의 전체 자원 수가 고정적이어야 함
- 자원이 낭비됨
- **항상 불안정 상태 방지해야 하므로 자원 이용도 낮음**

4 교착 상태 검출(탐지)

교착 상태 검출(탐지)의 개념

- 운영체제가 프로세스의 작업을 관찰하면서 교착 상태 발생 여부를 계속 주시하는 방식
- 교착 상태가 발견되면 이를 해결하기 위해 교착 상태 회복 단계를 밟음

타임아웃을 이용한 교착 상태 검출

- 일정 시간 동안 작업이 진행되지 않은 프로세스를 교착 상태가 발생한 것으로 간주하여 처리하는 방법
- 교착 상태가 자주 발생하지 않을 것이라는 가정하에 사용하는 것으로, 특별한 알고리즘이 없어 쉽게 구현할 수 있음
- 타임아웃이 되면 프로세스가 종료됨

그림 6-19 타임아웃을 이용한 방법의 예

4 교착 상태 검출

데이터베이스에서 타임아웃의 문제

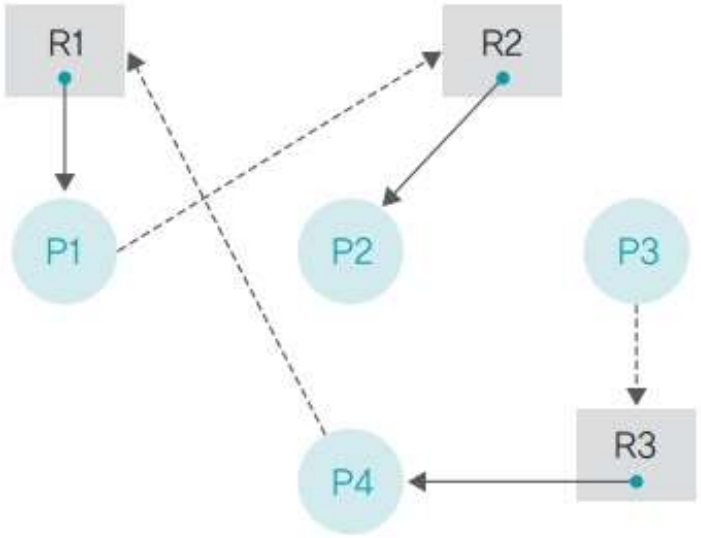
- 데이터베이스에서 타임아웃으로 프로세스가 종료되면 일부 데이터의 일관성이 깨질 수 있음
- 데이터의 일관성이 깨지는 문제를 해결하기 위해 체크포인트와 롤백 사용
  - 체크포인트 : 작업을 하다가 문제가 발생하면 저장된 상태로 돌아오기 위한 표시
  - 롤백 : 작업을 하다가 문제가 발생하여 과거의 체크포인트로 되돌아가는 것

그림 6-20 체크포인트와 롤백

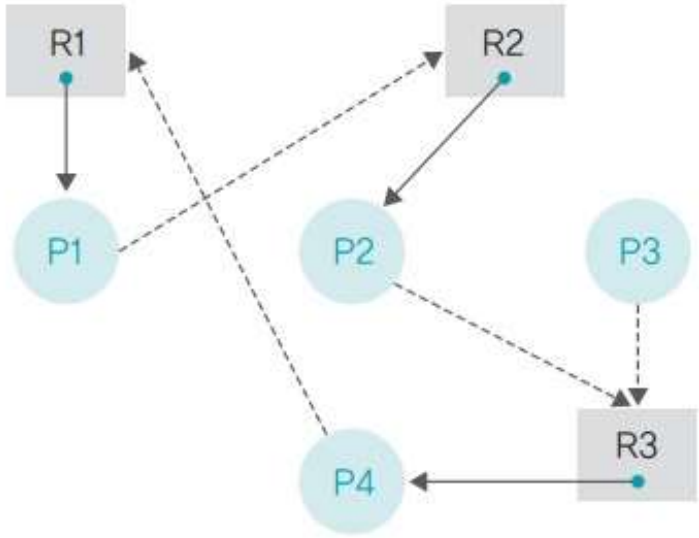
4 교착 상태 검출

자원 할당 그래프를 이용한 교착 상태 검출

- 단일 자원을 사용하는 경우 자원 할당 그래프에 사이클 있으면 교착 상태



(a) 교착 상태가 없는 자원 할당 그래프



(b) 교착 상태가 있는 자원 할당 그래프

그림 6-22 **자원 할당 그래프와 교착 상태**

3. 교착 상태 탐지

교착 상태 탐지 알고리즘

- 쇼사니Shoshani와 코프만Coffman이 제안

- **은행가 알고리즘에서 사용한 자료구조들과 비슷**
  - **Available : 자원마다 사용 가능한 자원 수를 표시하는 길이 m 인 벡터**
  - **Allocation : 각 프로세스에 현재 할당된 각 형태들의 자원 수 표시하는n ×m 행렬**
  - **Request : 각 프로세스의 현재 요청 표시하는 n×m 행렬. Request[i, j]일 때 프로세스 Pi에 필요한 자원 수가 k개라면, 프로세스 Pi는 자원 Ri의 자원 k개 더 요청**

3. 교착 상태 탐지

- 교착 상태 탐지 알고리즘 순서 **탐지 알고리즘은 남아 있는 프로세스들의 할당 가능 순서 모두 찾음**

  - 1단계 : Work와 Finish는 각각 길이가 m과 n인 벡터이다.

    Work = Available로 초기화한다. (i = 1, 2, ……, n)일 때 $Allocation_i \neq 0$이면

    Finish[i] = false이고, 아니면 Finish[i] = true이다.

  - 2단계 : 다음 조건을 만족하는 색인 i를 찾는다.

    Finish[i] == false

    $Request_i \leq Work$

    조건에 맞는 i가 없으면 4단계로 이동한다.

  - 3단계 : 다음 조건과 일치하는지 여부를 판단하여 2단계로 이동한다.

    $Work = Work + Allocation_i$

    Finish[i] = true

  - 4단계 : Finish[i] == false라면, $1 \leq i \leq n$인 범위에서 시스템은 교착 상태에 있다.

    또 프로세스 P_i도 교착 상태에 있다.

그림 5-19 교착 상태 탐지 알고리즘

3. 교착 상태 탐지

- 시간 t_0일 때 시스템의 상태 현재 이 시스템은 교착 상태에 있지 않음

표 5-1 시간 t_0일 때 시스템의 상태

프로세스	Allocation	Request	Available
	ABC	ABC	ABC
P_0	010	000	000
P_1	200	202	
P_2	303	000	
P_3	211	100	
P_4	002	002	

- 1단계 : Work와 Finish는 각각 길이가 m과 n인 벡터이다.
  Work = Available로 초기화한다. (i = 1, 2, ……, n)일 때 $Allocation_i \neq 0$이면 Finish[i] = false이고, 아니면 Finish[i] = true이다.
- 2단계 : 다음 조건을 만족하는 색인 i를 찾는다.
  Finish[i] == false
  $Request_i \leq Work$
  조건에 맞는 i가 없으면 4단계로 이동한다.
- 3단계 : 다음 조건과 일치하는지 여부를 판단하여 2단계로 이동한다.
  $Work = Work + Allocation_i$
  Finish[i] = true
- 4단계 : Finish[i] == false라면, $1 \leq i \leq n$인 범위에서 시스템은 교착 상태에 있다.
  또 프로세스 P_i도 교착 상태에 있다.

그림 5-19 교착 상태 탐지 알고리즘

3. 교착 상태 탐지

- 프로세스 P_2가 자원 C의 자원을 1개 더 요청할때 현재 이 시스템은 교착 상태

표 5-2 시간 t_0일 때 시스템의 새로운 상태

프로세스	Request
	ABC
P_0	000
P_1	202
P_2	001
P_3	100
P_4	002

- 1단계 : Work와 Finish는 각각 길이가 m과 n인 벡터이다.

  Work = Available로 초기화한다. (i = 1, 2, ……, n)일 때 $Allocation_i \neq 0$이면 Finish[i] = false이고, 아니면 Finish[i] = true이다.
- 2단계 : 다음 조건을 만족하는 색인 i를 찾는다.

  Finish[i] == false

  $Request_i \leq Work$

  조건에 맞는 i가 없으면 4단계로 이동한다.
- 3단계 : 다음 조건과 일치하는지 여부를 판단하여 2단계로 이동한다.

  $Work = Work + Allocation_i$

  Finish[i] = true
- 4단계 : Finish[i] == false라면, $1 \leq i \leq n$인 범위에서 시스템은 교착 상태에 있다.

  또 프로세스 P_i도 교착 상태에 있다.

그림 5-19 교착 상태 탐지 알고리즘

5 교착 상태 회복

교착 상태 회복

- 교착 상태가 검출된 후 교착 상태를 푸는 후속 작업을 하는 것
- 교착 상태 회복 단계에서는 교착 상태를 유발한 프로세스를 강제로 종료
- 프로세스를 강제로 종료하는 방법
  - ❶ 교착 상태를 일으킨 모든 프로세스를 동시에 종료
  - ❷ 교착 상태를 일으킨 프로세스 중 하나를 골라 순서대로 종료
    - ✓ 우선순위가 낮은 프로세스를 먼저 종료
    - ✓ 우선순위가 같은 경우 작업 시간이 짧은 프로세스를 먼저 종료
    - ✓ 위의 두 조건이 같은 경우 자원을 많이 사용하는 프로세스를 먼저 종료
  - **최소 비용으로 프로세스들을 중단하는 우선 순위 선정**
    - ✓ 프로세스가 수행된 시간과 앞으로 종료하는 데 필요한 시간
    - ✓ 프로세스가 사용한 자원 형태와 수(예 : 자원을 선점할 수 있는지 여부)
    - ✓ 프로세스를 종료하는 데 필요한 자원 수
    - ✓ 프로세스를 종료하는 데 필요한 프로세스 수
    - ✓ 프로세스가 대화식인지, 일괄식인지 여부

Homework

6장 연습문제

- P.323 연습문제(단답형)
- P.324 심화문제(서술형)
- 제출기한 : 11월 8일(일) 23시
- 제출방법 : hwp/word로 작성한 후 eclass 과제함
- 파일이름 : **학번(이름)-6장**. pdf

Thank You

1 다중 자원과 사이클

다중 자원과 사이클

- 다중 자원이 포함된 자원 할당 그래프에서는 대기 그래프와 그래프 감소 방법을 이용하여 사이클을 찾음

대기 그래프

- 자원 할당 그래프에서 프로세스와 프로세스 간에 기다리는 관계만 나타낸 그래프

그래프 감소

- 대기 그래프에서 작업이 끝날 가능성이 있는 프로세스의 화살표와 관련 프로세스의 화살표를 연속적으로 지워가는 작업

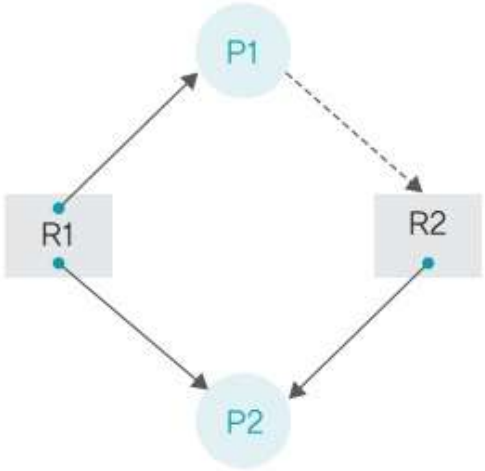
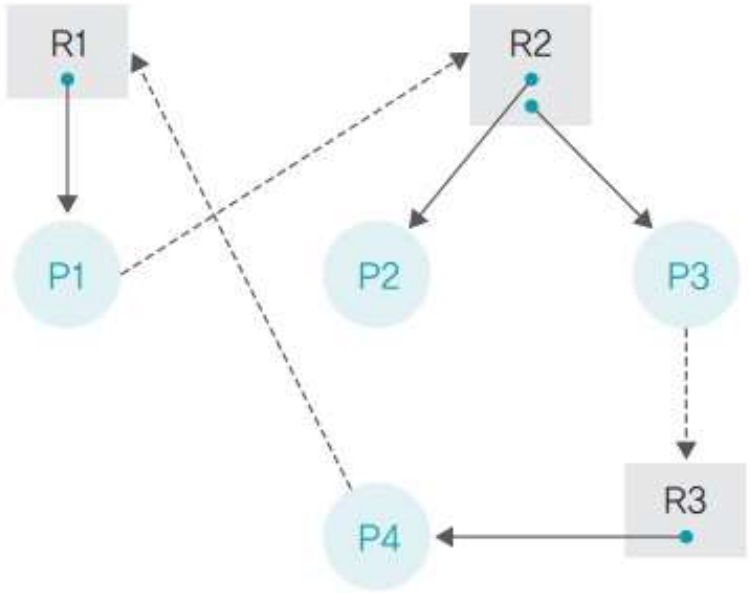


그림 6-23 다중 자원과 사이클

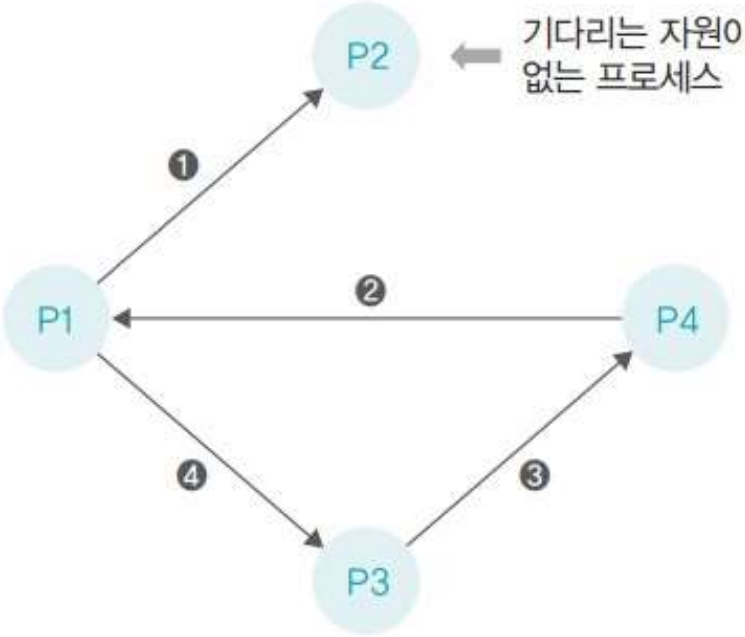
2 대기 그래프와 그래프 감소

다중 자원 사용 시 교착 상태가 발생하지 않는 경우

- 그래프 감소 결과 사이클이 남아 있지 않으므로 교착 상태가 발생하지 않음



(a) 자원 할당 그래프



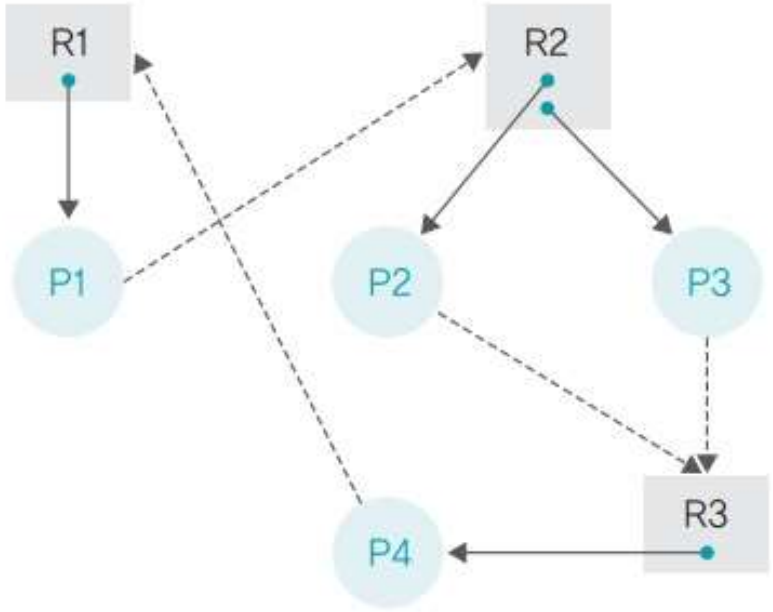
(b) 대기 그래프

그림 6-24 다중 자원 사용 시 교착 상태가 발생하지 않는 경우

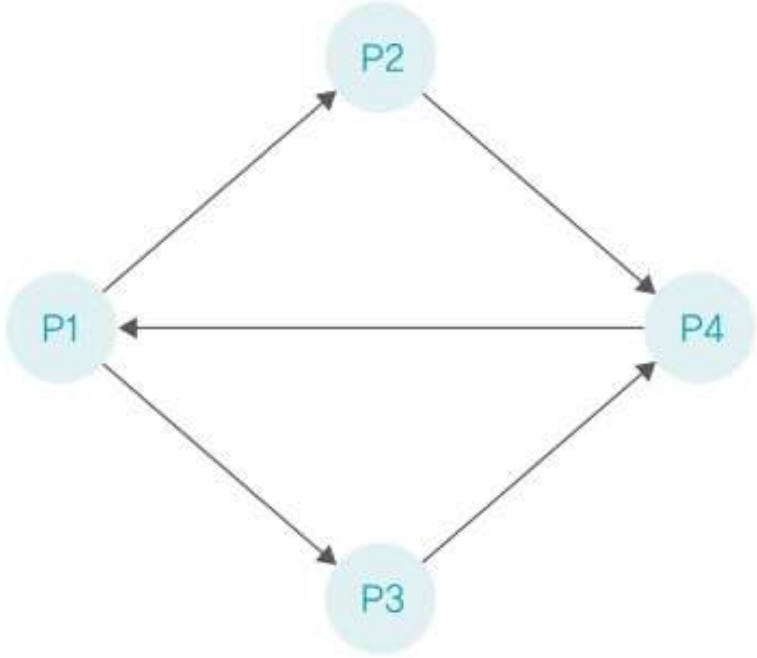
2 대기 그래프와 그래프 감소

다중 자원 지원 시 교착 상태가 발생하는 경우

- 그래프 감소를 해도 여전히 사이클이 남아 있어 교착 상태가 발생



(a) 자원 할당 그래프



(b) 대기 그래프

그림 6-25 다중 자원 사용 시 교착 상태가 발생하는 경우