# 운영체제의 개요

# 1-1 일상생활 속의 운영체제

## 운영체제(OS, Operating System)

- 일반 컴퓨터, 노트북, 스마트폰의 전원을 켜면 가장 먼저 만나게 되는 소프트웨어
- [예] PC 운영체제(윈도우, Mac OS, 유닉스, 리눅스 등)
  모바일 운영체제(iOS, 안드로이드 등)

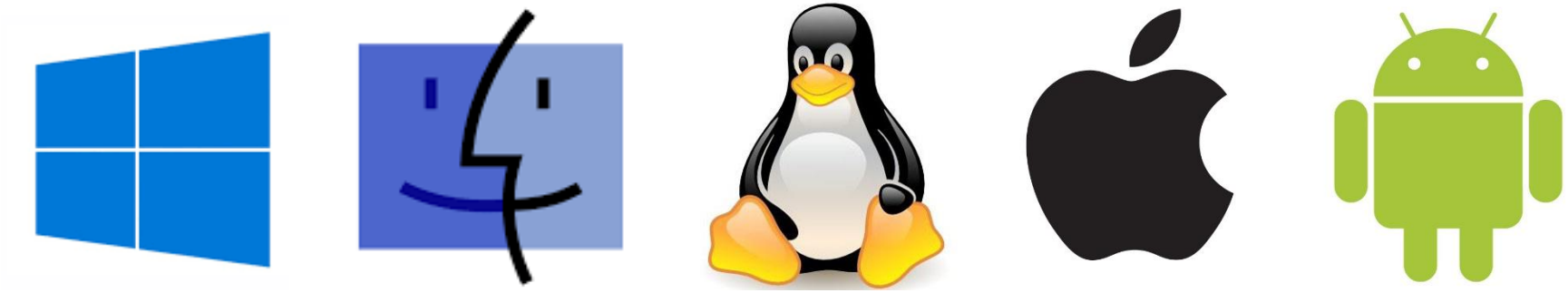


**그림 1-1** 다양한 운영체제(왼쪽부터 윈도우, Mac OS, 리눅스, iOS, 안드로이드)

# 1-1 일상생활 속의 운영체제

## 임베디드 운영체제

- CPU의 성능이 낮고 메모리 크기도 작은 시스템에 내장하도록 만든 운영체제
- 임베디드 운영체제가 있는 기계는 기능을 계속 향상할 수 있음


**그림 1-2** 임베디드 운영체제를 사용하는 스마트 시계와 스마트 TV

# 1-3 운영체제의 정의

## 운영체제의 정의

- 응용 프로그램이나 사용자에게 컴퓨터 자원을 사용할 수 있는 인터페이스를 제공하고 그 결과를 돌려주는 시스템 소프트웨어
- 응용 프로그램이나 사용자에게 모든 컴퓨터 자원을 숨기고 정해진 방법으로만 컴퓨터 자원을 사용할 수 있도록 제한


그림 1-4 운영체제의 정의

# 1-4 운영체제의 역할

## 자원 관리

- 컴퓨터 시스템의 자원을 응용 프로그램에 나누어주어 사용자가 원활하게 작업할 수 있도록 함
- 자원을 요청한 프로그램이 여러 개라면 적당한 순서로 자원을 배분하고 적절한 시점에 자원을 회수하여 다른 응용 프로그램에 나누어줌

## 자원 보호

- 비정상적인 작업으로부터 컴퓨터 자원을 보호

## 하드웨어 인터페이스 제공

- 사용자가 복잡한 과정 없이 다양한 장치를 사용할 수 있도록 해주는 하드웨어 인터페이스 제공
- CPU, 메모리, 키보드, 마우스와 같은 다양한 하드웨어를 일관된 방법으로 사용할 수 있도록 지원

## 사용자 인터페이스 제공

- 사용자가 운영체제를 편리하게 사용하도록 지원(예: 윈도우의 그래픽 사용자 인터페이스(GUI))

# 교착 상태

# 1-1 교착 상태의 정의

## 교착 상태

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

## 아사 상태와 차이점

- **아사 현상** : 운영체제가 잘못된 정책을 사용하여 특정 프로세스의 작업이 지연되는 문제
- **교착 상태** : 여러 프로세스가 작업을 진행하다보니 자연 발생적으로 일어나는 문제

그림 6-2 요리사의 자원 선점과 교착 상태

# 1-2 교착 상태의 발생

## 시스템 자원

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

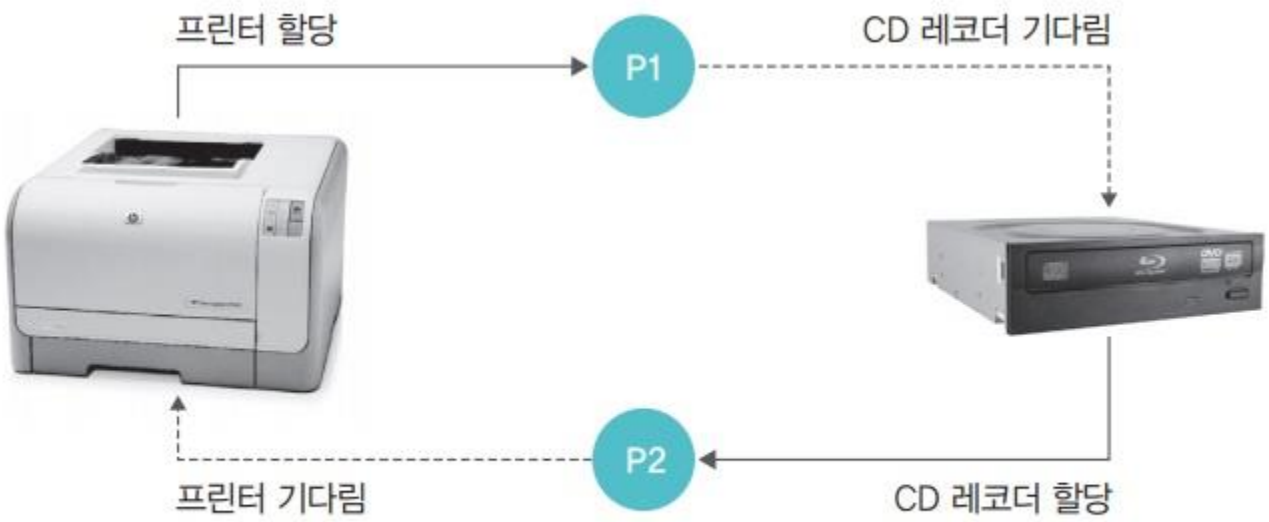



그림 6-3 시스템 자원 사용 도중 교착 상태 발생

# 1-2 교착 상태의 발생

## 공유 변수

- 교착 상태는 공유 변수를 사용할 때 발생

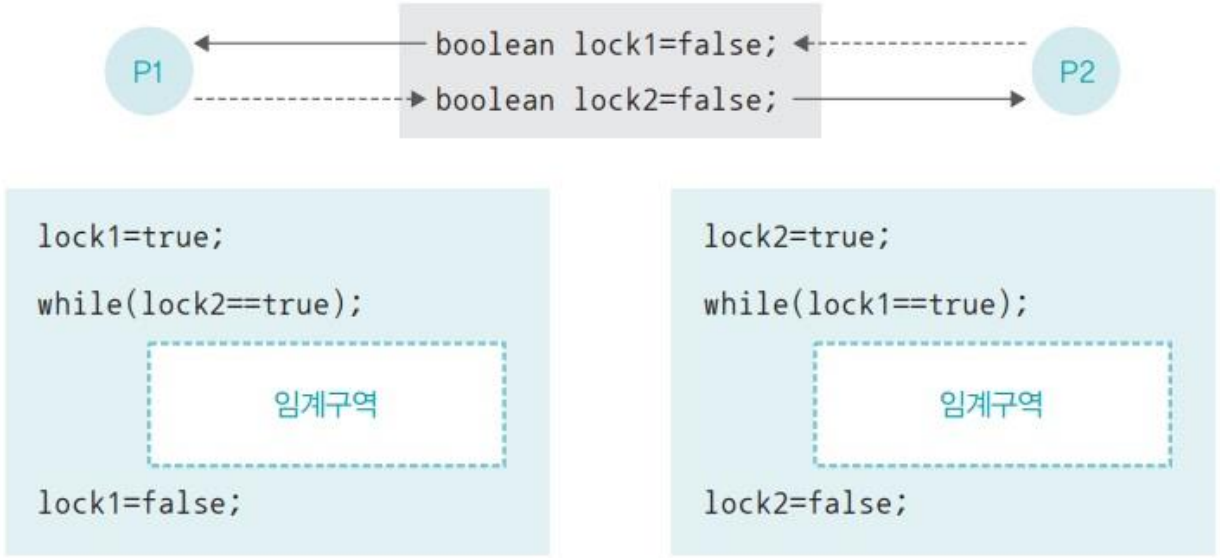



그림 6-4 교착 상태를 발생시키는 임계구역 코드

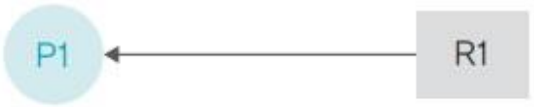
## 응용 프로그램

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

# 1-3 자원 할당 그래프

## 자원 할당 그래프

- 프로세스가 어떤 자원을 사용 중이고 어떤 자원을 기다리고 있는지를 방향성이 있는 그래프로 표현한 것
- 프로세스는 원으로, 자원은 사각형으로 표현




프로세스 P1이 자원 R1을 할당받음


프로세스 P1이 자원 R1을 기다림

그림 6-5 **자원 할당 그래프**

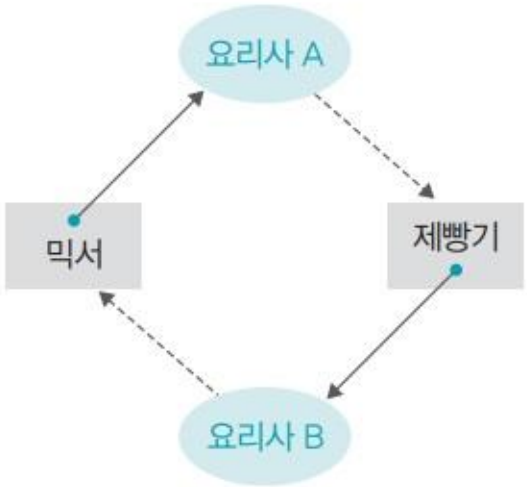



그림 6-7 **요리사 문제의 자원 할당 그래프**

# 1-3 자원 할당 그래프

## 다중 자원

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

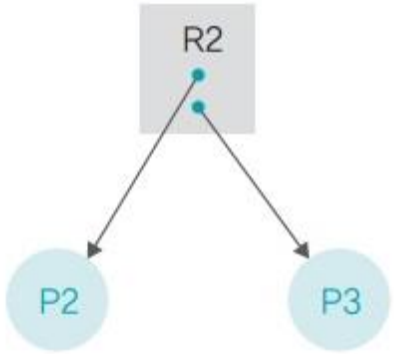



그림 6-6 2개의 프로세스를 수용할 수 있는 자원

# 1-3 자원 할당 그래프

## 식사하는 철학자 문제

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

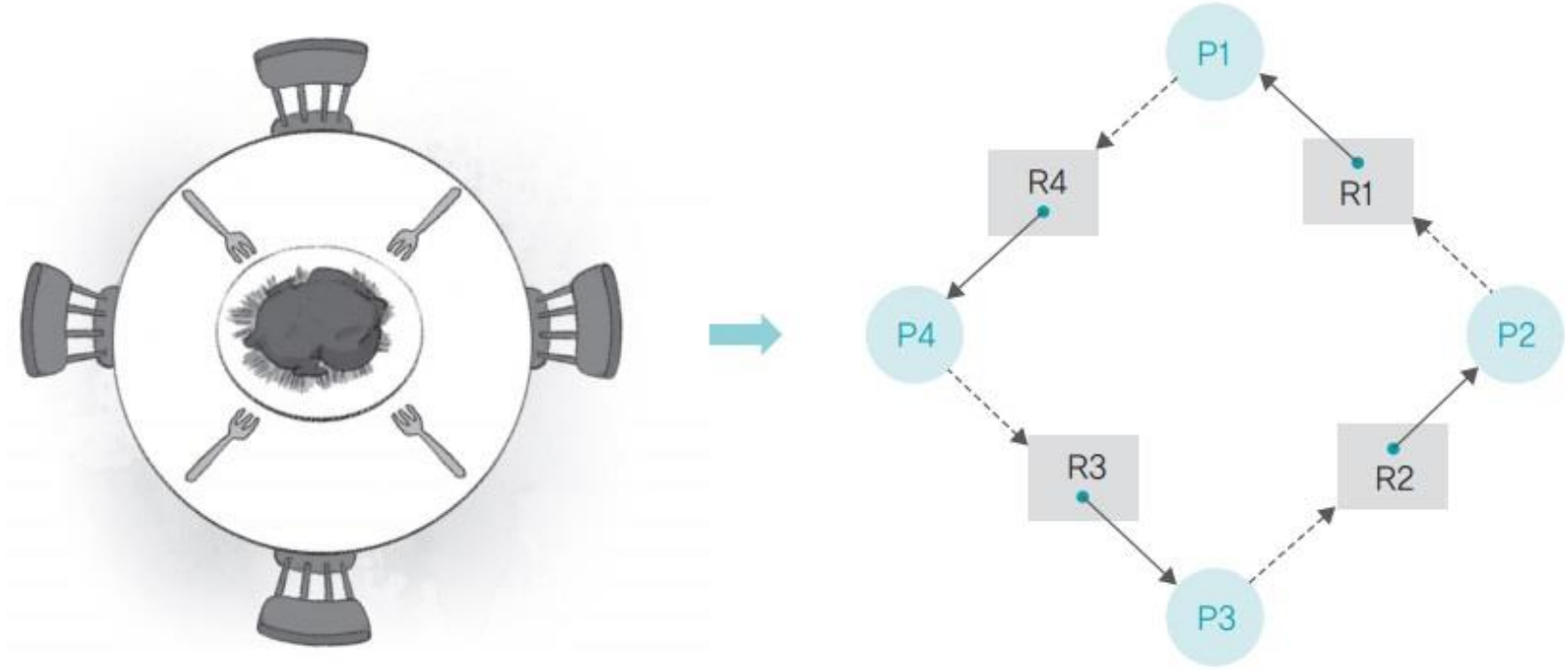



그림 6-8 식사하는 철학자 문제의 자원 할당 그래프

# 1-3 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

# 2-1 교착 상태 필요조건

## 교착 상태 필요 조건

- **다음 4가지 조건이 모두 발생해야만 교착상태 발생**(필요조건)
  - 4가지 중 단 한가지라도 만족하지 않으면 교착상태가 발생하지 않음

- **상호 배제**(mutual exclusion) : 한 프로세스가 사용하는 자원은 다른 프로세스와 공유할 수 없는 배타적인 자원이어야 함

- **비선점**(non-preemptive) : 한 프로세스가 사용 중인 자원은 중간에 다른 프로세스가 빼앗을 수 없는 비선점 자원이어야 함

- **점유와 대기**(hold and wait) : 프로세스가 어떤 자원을 할당받은 상태에서 다른 자원을 기다리는 상태여야 함

- **원형 대기**(circular wait) : 점유와 대기를 하는 프로세스 간의 관계가 원을 이루어야 함

# 2-1 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.

# 2-2 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)

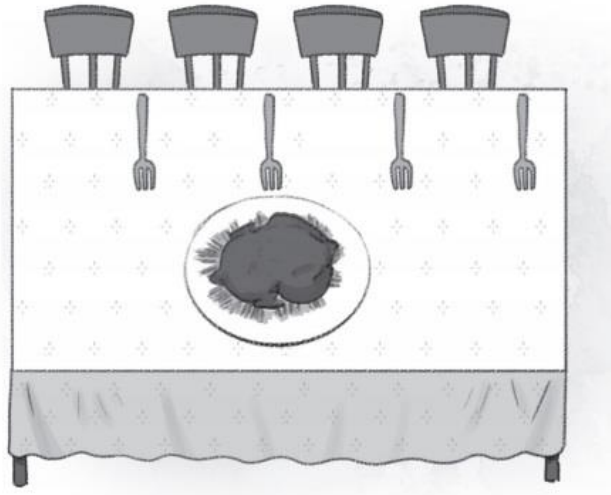


그림 6-10 사각형 식탁과 식사하는 철학자 문제

# 3-1 교착 상태 해결 방법

## 교착 상태 해결 방법

- **교착 상태 예방**(prevention) : 교착 상태를 유발하는 네 가지 조건이 발생하지 않도록 무력화하는 방식으로 교착상태 조선 4가지에 대하여 각각의 방법이 존재 함.

- **교착 상태 회피**(avoidance) : 교착상태가 발생하지 않도록 자원 할당량을 조절하여 교착 상태를 회피하는 방식

- **교착 상태 검출**(detection)**과 회복**(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

# 3-2 교착 상태 예방

## 교착상태 조건 4가지에 대하여 각각의 방식이 존재

## 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

# 3-2 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

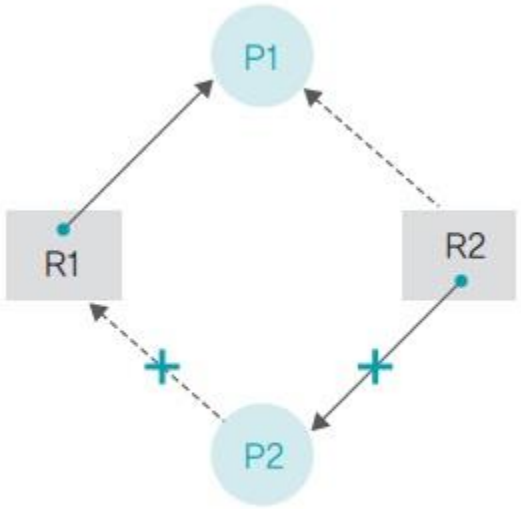



그림 6-11 전부 할당하거나 아니면 아예 할당하지 않는 방식

# 3-2 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

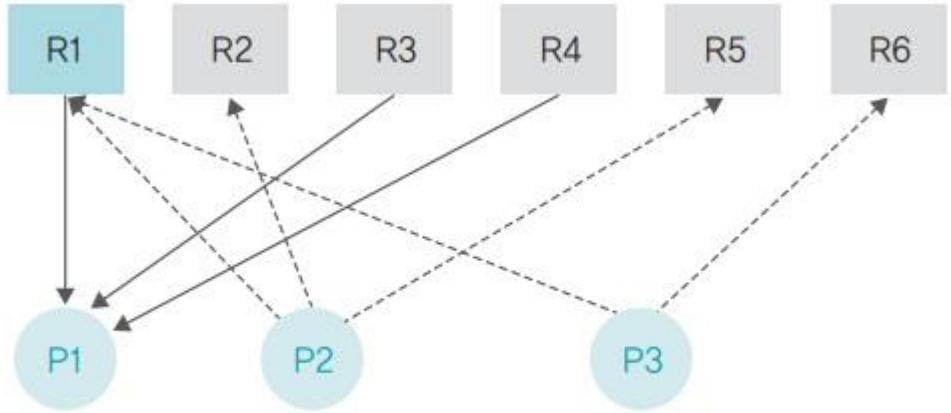



그림 6-12 필요한 자원을 모두 할당한 후 실행

# 3-2 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

# 3-2 교착 상태 예방

## 원형 대기 예방과 교착 상태 해결

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

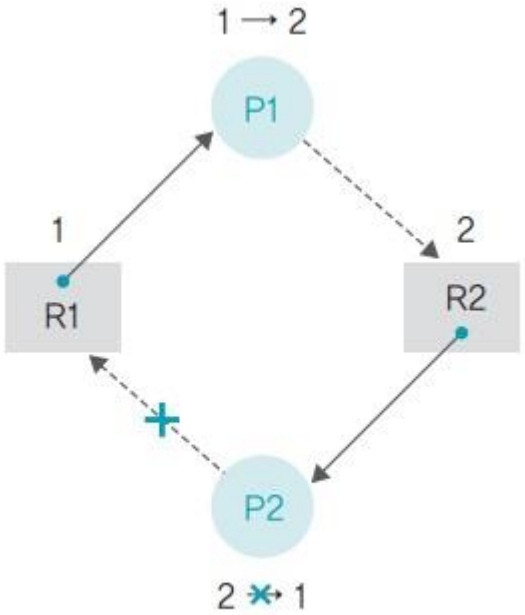



그림 6-14 원형 대기 조건의 부정

## 원형 대기 예방의 단점

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

# 3-2 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

# 3-3 교착 상태 회피

## 교착 상태 회피의 개념

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

# 3-3 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

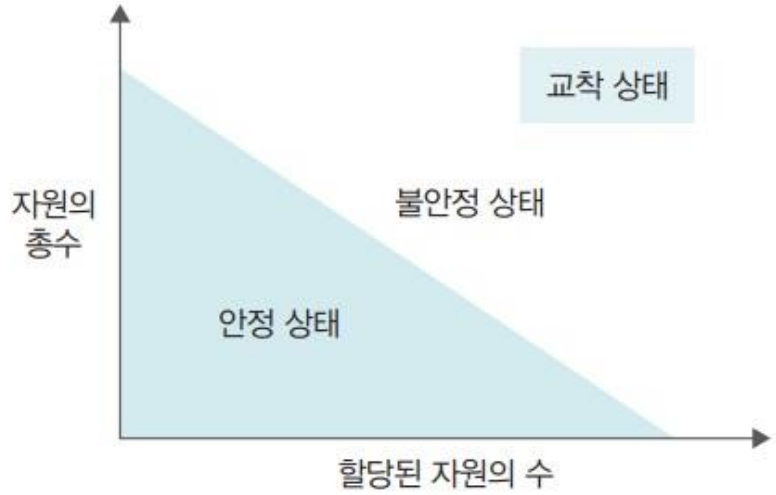



그림 6-15 안정 상태와 불안정 상태

# 3-3 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

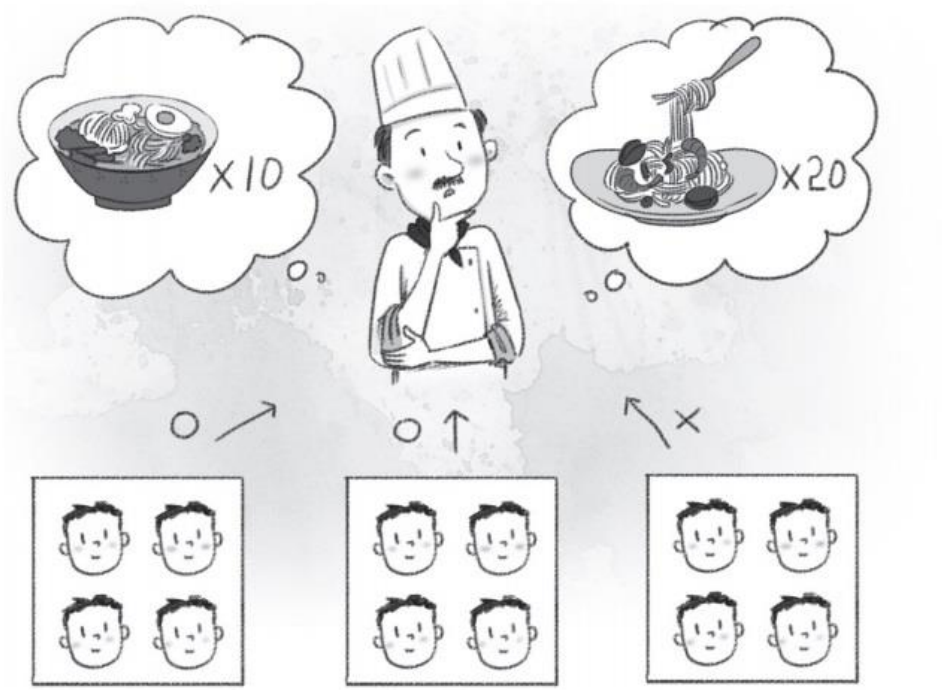



그림 6-16 은행원 알고리즘

# 3-3 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) |

# 3-3 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.

# 3-3 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)

# 3-3 교착 상태 회피

## 교착 상태 회피의 문제점

- 프로세스가 자신이 사용할 모든 자원을 미리 선언해야 함
- 시스템의 전체 자원 수가 고정적이어야 함
- 자원이 낭비됨

# 3-4 교착 상태 검출

## 교착 상태 검출의 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

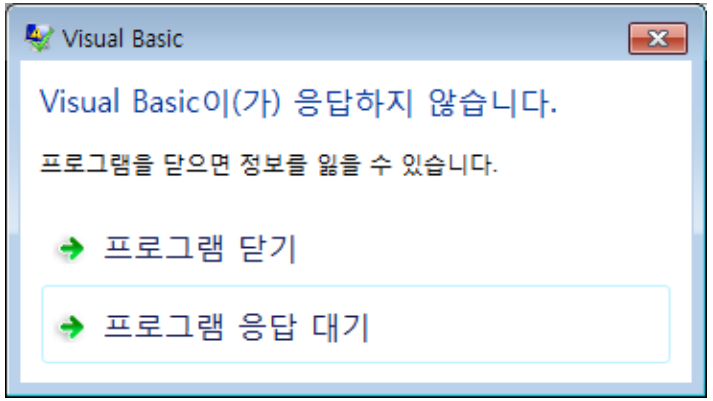



**그림 6-19** 타임아웃을 이용한 방법의 예

# 3-4 교착 상태 검출

## 데이터베이스에서 타임아웃의 문제

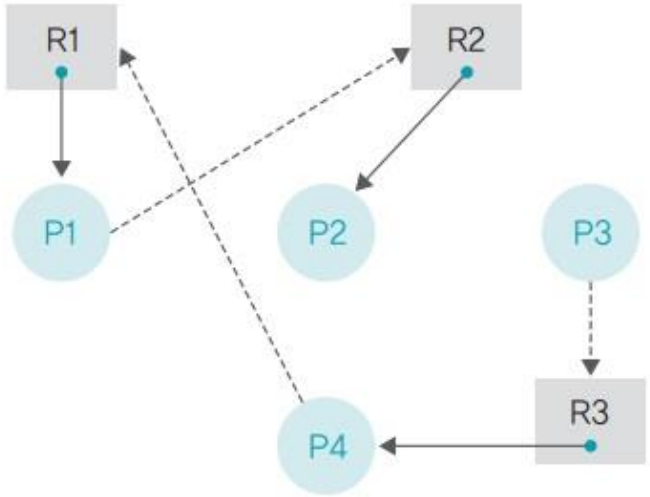
- 데이터베이스에서 타임아웃으로 프로세스가 종료되면 일부 데이터의 일관성이 깨질 수 있음
- 데이터의 일관성이 깨지는 문제를 해결하기 위해 체크포인트와 롤백 사용
  - **체크포인트** : 작업을 하다가 문제가 발생하면 저장된 상태로 돌아오기 위한 표시
  - **롤백** : 작업을 하다가 문제가 발생하여 과거의 체크포인트로 되돌아가는 것


그림 6-20 **체크포인트와 롤백**

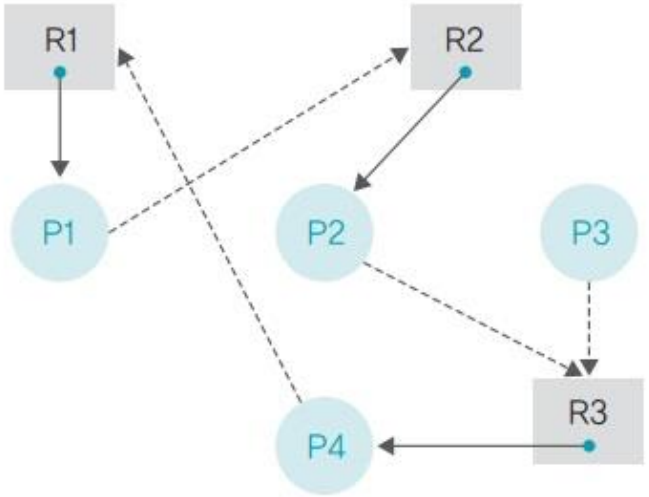
# 3-4 교착 상태 검출

## 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**

# 3-5 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