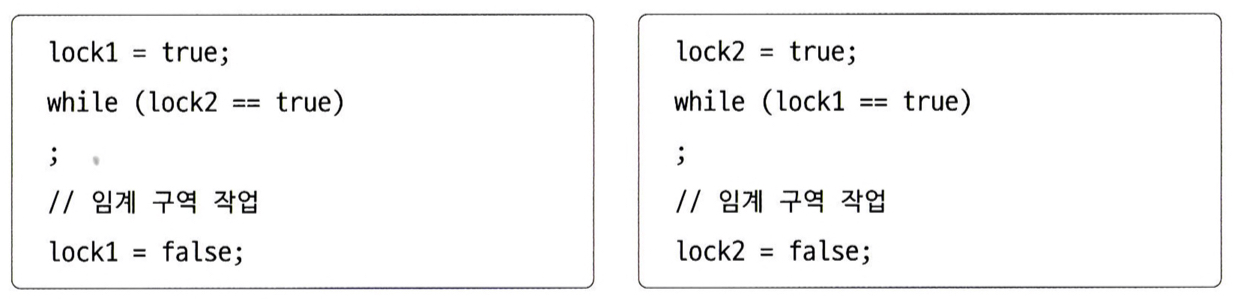
**Chapter 13 교착 상태**

***13-1*** **교착 상태란**

**교착 상태**: 프로세스 실행을 위해 필요한 자원을 두 개 이상의 프로세스가 무작정 기다리며 진행이 멈춰 버리는 현상 (**식사하는 철학자 문제**)



**<자원 할당 그래프>**

**자원 할당 그래프**: 어떤 프로세스가 어떤 자원을 사용하고 있고, 어떤 프로세스가 어떤 자원을 기다리고 있는지를 표현하는 간단한 그래프. 교착 상태를 단순하게 표현 가능

* **프로세스**는 **원**으로, **자원의 종류**는 **사각형**으로 표현



* 사용할 수 있는 자원의 개수는 자원 사각형 내에 점으로 표현



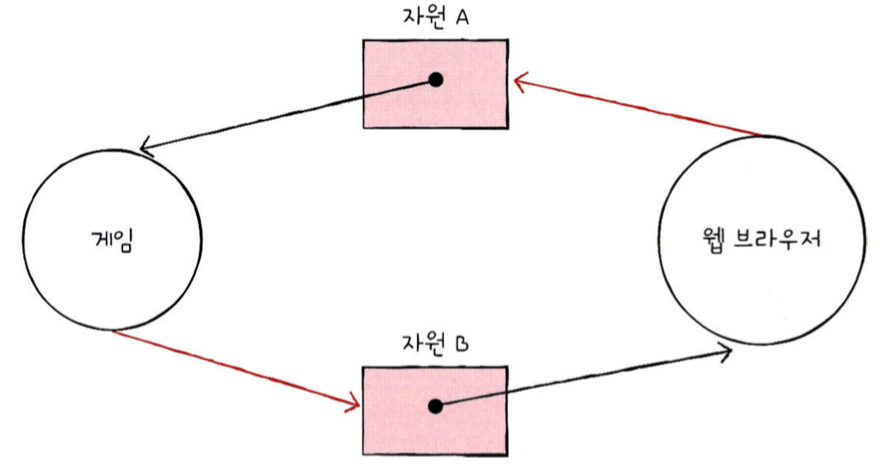
* 프로세스가 어떤 자원을 할당받아 사용 중이라면 프로세스를 향해 화살표를 표시



* 프로세스가 어떤 자원을 기다리고 있다면 프로세스에서 자원으로 화살표를 표시



* 교착 상태가 발생한 상황은 자원 할당 그래프가 원의 형태를 띄고 있음



**<교착 상태 발생 조건>**

교착 상태가 발생하는 조건에는 상호배제, 점유와 대기, 비선점, 원형 대기가 있음. 네 가지 조건이 모두 만족될 때 교착 상태가 발생한 가능성이 생김

**상호 배제:** 한 프로세스가 사용하는 자원을 다른 프로세스가 사용할 수 없는 상황

**점유와 대기**:어떠한 자원을 할당받은 상태에서 다른 자원을 할당받기를 기다리는 상태

**비선점**: 비선점 자원은 그 자원을 이용하는 프로세스의 작업이 끝나야만 비로소 이용 가능

(강제로 자원을 빼앗지 못함)

**원형 대기**: 자원 할당 그래프가 원의 형태로 그려지는 상태

***13-2*** **교착 상태 해결 방법**

교착 상태를 해결하는 방법은 예방, 회피, 검출 후 회복이 있음

**<교착 상태 예방>**

교착 상태를 예방하는 방법은 교착 상태 발생 필요 조건 네 가지 중 하나를 충족하지 못하게 하는 방법

**상호 배제 제거**

* 모든 자원을 공유 가능하게 한다
* 이론적으로는 교착 상태를 없앨 수 있지만 현실적으로 모든 자원의 상호 배제를 없애기는 어려움

**점유와 대기 제거**

* 운영체제는 특정 프로세스에 자원을 모두 할당하거나, 아예 할당하지 않는 방식으로 배분
* 이론적으로는 교착 상태를 해결할 수 있지만 자원의 활용률이 낮아질 우려 존재
* 점유와 대기를 금지하면 많은 자원을 사용하는 프로세스가 불리해짐(기아현상)

-> 자원을 적게 사용하는 프로세스에 비해 동시에 자원을 사용할 타이밍을 확보하기 어렵기 때문

**비선점 조건 제거**

* 자원을 이용 중인 프로세스로부터 해당 자원을 빼앗을 수 있음
* 선점하여 사용할 수 있는 일부 자원에 대해서는 효과적 ex) CPU
* 선점할 수 없는 자원도 존재 ex) 프린터

**원형 대기 조건 제거**

* 모든 자원에 번호를 붙이고, 오름차순으로 자원을 할당하면 원형 대기 발생하지 않음
* 앞선 세 방식보다 비교적 현실적이고 실용적이지만 모든 자원에 번호를 붙이기는 쉽지 않고 각 자원에 어떤 번호를 붙이는지에 따라 특정 자원의 활용률이 떨어질 수 있음

교착상태 예방 방식은 교착 상태가 발생하지 않음을 보장할 수는 있지만 여러 부작용이 따름

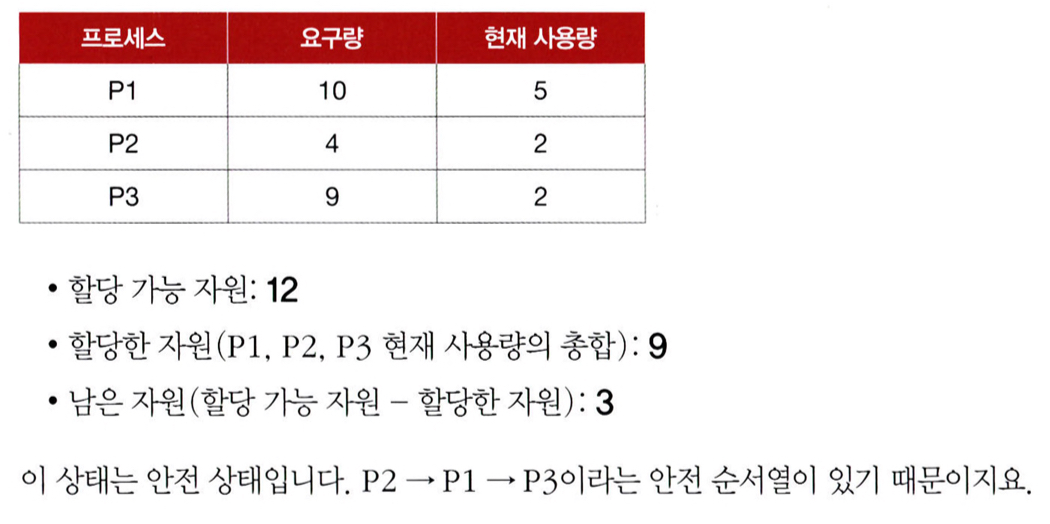
**<교착 상태 회피>**

교착 상태 회피는 프로세스들에 배분할 수 있는 자원의 양을 고려하여 교착 상태가 발생하지 않을 정도의 양만큼만 자원을 배분하는 방법

**안전 상태**: 교착 상태가 발생하지 않고 모든 프로세스가 정상적으로 자원을 할당받고 종료될 수 있는 상태 (**안전 순서열**이 **있는** 상태)

**불안정 상태**: 교착 상태가 발생할 수도 있는 상황(**안전 순서열**이 **없는** 상태)

**안전 순서열**: 교착 상태 없이 안전하게 프로세스들에 자원을 할당할 수 있는 순서



**안전 상태**



**불안전 상태**

교착 상태 회피 방식은 항시 안전 상태를 유지하도록 자원을 할당하는 방식

**<교착 상태 검출 후 회복>**

교착 상태 검출 후 회복은 교착 상태 발생을 인정하고 사후에 조치하는 방식

운영체제는 프로세스들이 자원을 요구할 때마다 그때때 모두 할당하며 교착 상태 발생 여부를 주기적으로 검사. 교착 상태가 검출되면 그때 다음과 같은 방식으로 회복

**선점을 통한 회복**:교착 상태가 해결될 때까지 다른 프로세스로부터 자원을 빼앗아 한 프로세스씩 자원을 몰아주는 방식

**프로세스 강제 종료를 통한 회복**

* 가장 단순하면서 확실한 방식
* 운영체제는 교착 상태에 놓인 프로세스를 모두 강제 종료 가능

-> 한 방에 교착 상태를 해결할 수 있는 가장 확실한 방식이나 그만큼 많은 프로세스들이 작업 내역을 잃게 될 가능성 있음

* 교착 상태가 없어질 때까지 한 프로세스씩 강제 종료 가능

-> 작업 내역을 잃는 프로세스는 최대한 줄일 수 있지만 교착 상태가 없어졌는지 여부를 확인하는 과정에서 오버헤드를 야기

**타조 알고리즘**: 교착 상태를 아예 무시하는 방법. 문제 발생의 빈도나 심각성에 따라 최대 효율을 추구하는 엔지니어 입장에서는 때때로 이 방식이 적합할 때도 존재