**챕터8 Deadlocks; 교착 상태**

**8.3 교착상태 특성**

**8.3.1 프로세스**

교착 상태는 한 시스템에서 다음 네 가지 조건이 동시에 성립될 때 발생할 수 있다.

1. 상호 배제(mutual exclusion): 최소한 하나의 자원이 비공유 모드로 점유되어야 한다. 비공유 모드에서는 한 번에 한 스레드만이 그 자원을 사용할 수 있다. 다른 스레드가 그 자원을 요청하면, 요청 스레드는 자원이 방출될 때까지 반드시 지연되어야 한다.

2. 점유하며 대기(hold-and-wait): 스레드는 최소한 하나의 자원을 점유한 채 현재 다른 스레드에 의해 점유된 자원을 추가로 얻기 위해 반드시 대기해야 한다.

3. 비선점(no preemption): 자원들을 선점할 수 없어야 한다. 즉, 자원이 강제적으로 방출될 수 없고, 점유하고 있는 스레드가 태스크를 종료한 후, 그 스레드에 의해 자발적으로만 방출될 수 있다.

4. 순환 대기(circular wait): 대기하고 있는 스레드의 집합 {T0, T1, …, Tn}에서 T0는 T1이 점유한 자원을 대기하고, T1은 T2가 점유한 자원을 대기하고, …, Tn-1은 쑤이 점유한 자원을 대기하며, Tn은 T0가 점유한 자원을 대기한다.

**8.3.2 자원 할당 그래프**

이 그래프는 정점(vertex) V의 집합과 간선(edge) E의 집합으로 구성된다.

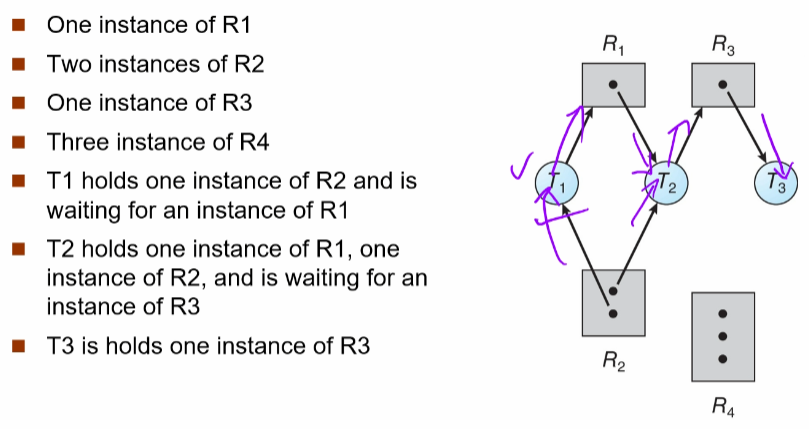
정점 V의 집합은 시스템 내의 모든 활성 스레드의 집합인 T = {T1, T2, …, Tn}과

시스템 내의 모든 자원 유형의 집합인 R = {R1, R2, …, Rm}의 두 가지 유형으로 구별된다.

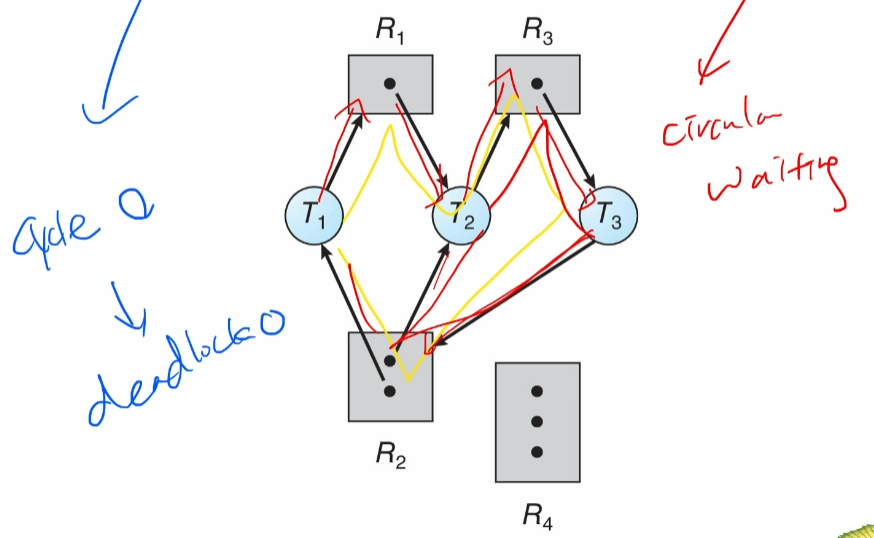
요청 간선(request edge): 스레드 Ti가 자원 유형 Rj의 인스턴스를 하나 요청하는 것으로 현재 이 자원을 기다리는 상태이다.

할당 간선(assignment edge): 자원 유형 Rj의 한 인스턴스가 스레드 Ti에 할당된 것을 의미한다.

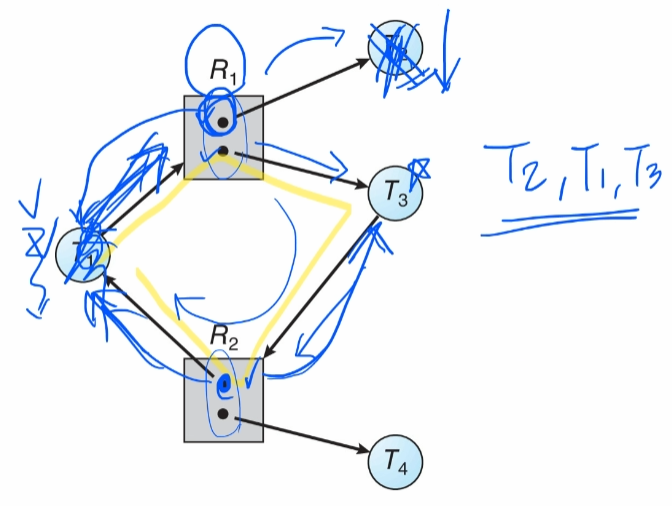
자원 할당 그래프



교착 상태를 갖는 자원 할당 그래프



사이클이 있으면서 교착 상태가 아닌 자원 할당 그래프



자원 할당 그래프에 사이클이 없다면, 시스템은 교착 상태가 아니다.

반면에, 사이클이 있다면 시스템은 교착 상태일 수도 있고 아닐 수도 있다.

**8.4 교착 상태 처리 방법**

원칙적으로 교착 상태 문제를 처리하는 데는 다음과 같은 세 가지 다른 방법이 있다.

1. 시스템이 결코 교착 상태가 되지 않도록 보장하기 위하여 교착 상태를 예방(Prevention)하거나 회피(Avoidance)하는 프로토콜을 사용한다.

2. 시스템이 교착 상태가 되도록 허용한 다음에 복구시키는 방법(Deadlock detection and recover)이 있다.

3. 문제를 무시(Ignorance)하고, 교착 상태가 시스템에서 절대 발생하지 않는 척한다.

**8.5 Deadlock Prevention; 교착 상태 예방**

교착 상태의 네 가지 조건 중 최소한 하나가 성립하지 않도록 보장함으로써, 우리는 교착 상태의 발생을 예방할 수 있다.

**8.5.1 Mutual Exclusion; 상호 배제**

상호 배제 조건이 성립되어야 한다. 즉 적어도 하나의 자원은 공유가 불가능한 자원이어야 한다.

반면에, 공유 가능한 자원들은 배타적인 접근을 요구하지 않으며, 따라서 교착 상태에 관련될 수 없다.

그러나 일반적으로 상호 배제 조건을 거부함으로써 교착 상태를 예방하는 것은 불가능하다.

어떤 자원들은 근본적으로 공유가 불가능하기 때문이다.

**8.5.2 Hold and Wait; 점유하며 대기**

시스템에서 점유하며 대기 조건이 발생하지 않도록 하려면 스레드가 자원을 요청할 때마다 다른 자원을 보유하지 않도록 보장해야 한다.

우리가 사용할 수 있는 하나의 프로토콜은 각 스레드가 실행을 시작하기 전에 모든 자원을 요청하고 할당해야 한다.

물론 이것은 자원 요청의 동적 특성으로 인해 대부분의 응용 프로그램에서는 실용적이지 않다.

한 대안 프로토콜은 스레드가 자원을 전혀 갖고 있지 않을 때만 자원을 요청할 수 있도록 허용한다. 스레드는 일부 자원을 요청하고 사용할 수도 있다. 스레드가 추가의 자원을 요청하려면, 자신에게 할당된 모든 자원을 반드시 먼저 방출해야 한다.

이 두 프로토콜은 두 가지 주요 단점이 있다.

첫째, 자원이 할당되었지만 장기간 사용되지 않을 수 있기 때문에 자원 이용률이 낮을 수 있다.

둘째, 기아가 발생할 수 있다. 인기 있는 여러 개의 자원이 필요한 스레드는 필요한 자원 중 적어도 하나는 항상 다른 스레드에 할당되므로 무한정 대기할 수 있다.

**8.5.3 No Preemption; 비선점**

만일 어떤 자원을 점유하고 있는 스레드가 즉시 할당할 수 없는 다른 자원을 요청하면(즉, 스레드가 반드시 대기해야 하면), 현재 점유하고 있는 모든 자원이 선점된다.

즉, 이들 자원들이 묵시적으로 방출된다.

선점된 자원들은 그 스레드가 기다리고 있는 자원들의 리스트에 추가된다.

스레드는 자신이 요청하고 있는 새로운 자원은 물론, 이미 점유하였던 옛 자원들을 다시 획득할 수 있을 때만 다시 시작될 것이다.

**8.5.4 Circular Wait; 순환 대기**

교착 상태 예방을 위해 지금까지 제시된 세 가지 옵션은 대부분 상황에서 일반적으로 실용적이지 않다.

그러나 교착 상태에 대한 네 번째이자 마지막 조건인 순환 대기 조건은 필요한 조건 중 하나를 무효화하여 실용적인 해결책을 제공할 수 있는 기회를 제공한다.

순환 대기 조건이 성립되지 않도록 하는 한 가지 방법은 모든 자원 유형에 전체적인 순서를 부여하여, 각 프로세스가 열거된 순서대로 오름차순으로 자원을 요청하도록 요구하는 것이다.

**8.6 Deadlock Avoidance; 교착 상태 회피**

교착 상태 회피 알고리즘은 시스템에 순환 대기 상황이 발생하지 않도록 자원 할당 상태를 검사한다.

**8.6.1 Safe State; 안전 상태**

시스템 상태가 안전(Safe)하다는 말은 시스템이 어떤 순서로든 스레드들이 요청하는 모든 자원을(최대 요구 수를 요구하더라도) 교착 상태를 야기시키지 않고 차례로 모두 할당해 줄 수 있다는 것을 뜻한다.

즉, 시스템이 안전 순서(safe sequence)를 찾을 수 있다면 시스템은 안전하다고 말한다.

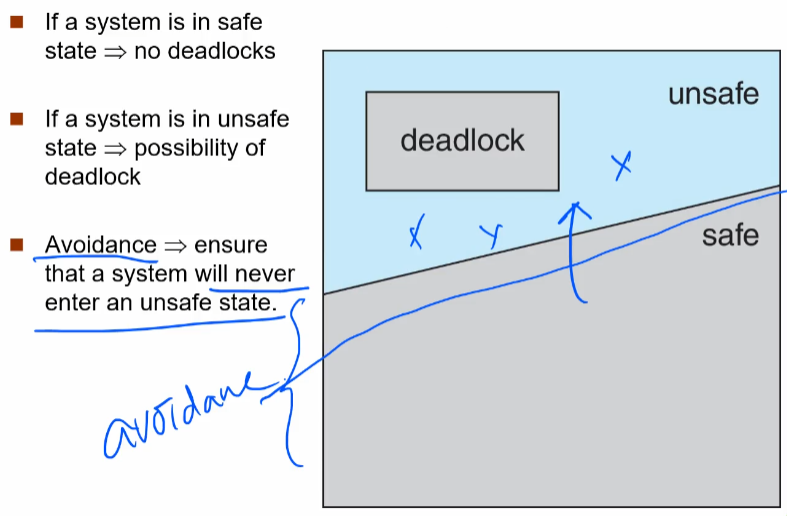
모든 스레드를 무사히 마칠 수 있는 시퀀스를 찾을 수 없으면 불안전(unsafe)하다고 한다.

시스템의 상태가 안전하다면 물론 교착 상태가 아니다. 반대로 교착 상태에 있는 시스템은 불안전한 상태에 있다.

그렇지만 시스템 상태가 불안전하다고 해서 반드시 교착 상태로 간다는 것을 뜻하는 것은 결코 아니다.

시스템이 불안전하다는 말은 앞으로 교착 상태로 가게 될 수도 있다는 뜻이다.

그러므로 시스템이 안전 상태에 머무는 한 운영체제는 불안전 상태나 교착 상태 모두를 예방할 수 있다.



**8.6.2 자원 할당 그래프 알고리즘**

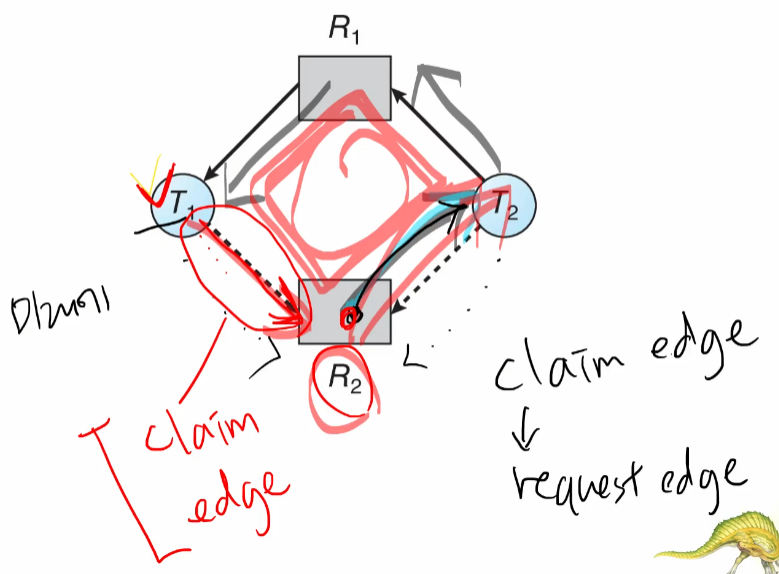
만약 각 자원 유형마다 단지 하나의 인스턴스를 갖는 자원 할당 시스템을 갖고 있다면, 우리는 교착 상태 회피를 위해 8.3.2절에서 정의한 자원 할당 그래프의 변형을 사용할 수 있다.

요청 간선과 할당 간선에 추가하여, 우리는 예약 간선(claim edge)이라는 새로운 타입의 간선을 도입한다.

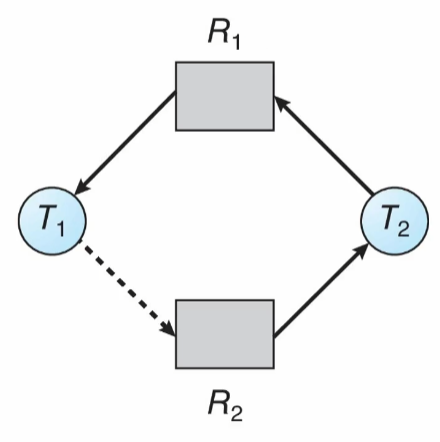
예약 간선 Ti -> Rj는 Ti가 미래에 자원 Rj를 요청할 것이라는 의미이다.

Ti가 자원 Rj를 요청하면, 예약 간선 Ti -> Rj는 요청 간선으로 변환된다.

교착 상태 회피를 위한 자원 할당 그래프

.

불안전 상태의 자원 할당 그래프



**8.6.3 Banker’s Algorithm; 은행원 알고리즘**

자원 할당 그래프 알고리즘은 종류마다 자원이 여러 개씩 있게 되면 사용할 수 없다. 아래에서 기술하는 알고리즘은 이와 같은 상황에서 사용할 수 있는 것이다.

하지만 앞의 자원 할당 그래프 알고리즘보다 아래 알고리즘은 효율성이 다소 떨어진다.

이 알고리즘은 통상 은행원(Banker’s) 알고리즘이라고 불린다.

이 시스템에서는 스레드가 시작할 때 스레드가 가지고 있어야 할 자원의 최대 개수를 자원 종류마다 미리 신고하여야 한다.

물론 이 숫자가 자원의 총 보유 수를 넘어서면 안 된다.

스레드가 자원들을 요청하면 시스템은 그것을 들어주었을 때 시스템이 계속 안전 상태에 머무르게 되는지 여부를 판단해야 한다.

계속 안전하게 된다면 그 요청을 들어준다. 그렇지 않다면 이 스레드의 요청은 허락이 안 된 채 다른 스레드가 끝날 때까지 기다리게 된다.

은행원 알고리즘을 구현하려면 몇 가지의 자료구조가 필요하다.

n이 스테드의 수이고 m이 자원의 종류 수라고 하자.

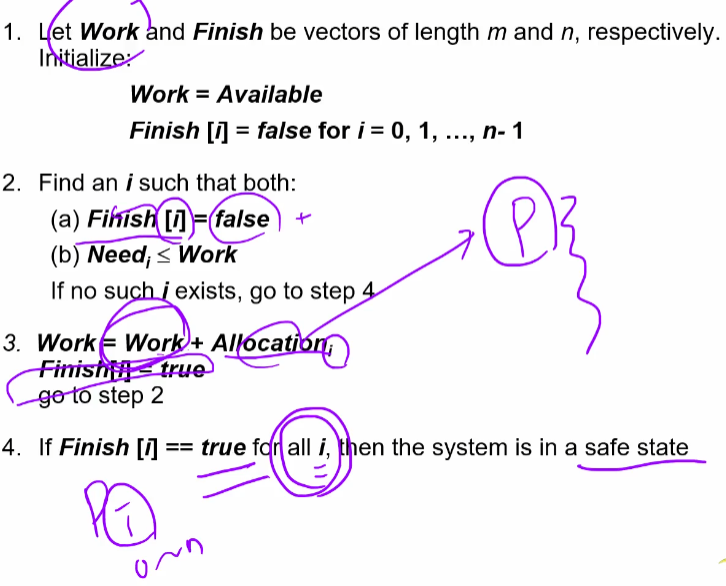
- **Available**: 각 종류별로 가용한 자원의 개수를 나타내는 벡터

- **Max**: 각 스레드가 최대로 필요로 하는 자원의 개수를 나타내는 행렬

- **Allocation**: 각 스레드에 현재 할당된 자원의 개수를 나타내는 행렬

- **Need**: 각 스레드가 향후 요청할 수 있는 자원애 개수를 나타내는 행렬

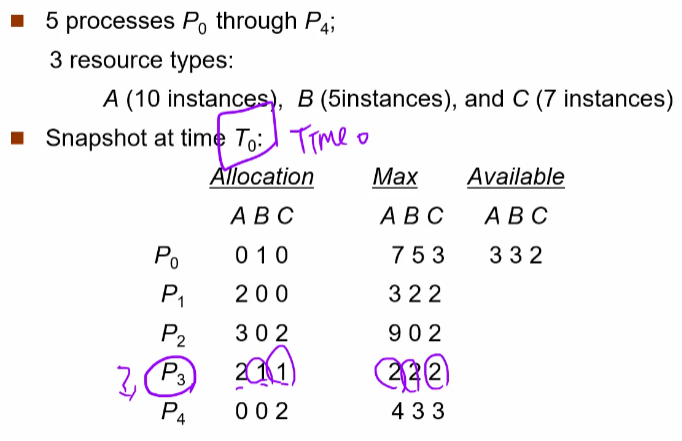
**8.6.3.1 Safety Algorithm; 안전성 알고리즘**



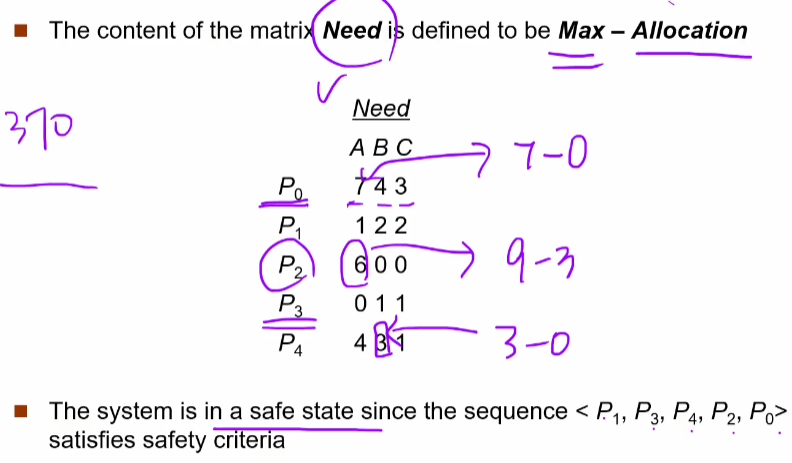
**8.6.3.3 예시**

다섯 개의 프로세스 T0부터 T4까지 있고, A, B, C 세 가지 종류의 자원이 있다고 가정하자.

시스템에는 A 자원이 10개, B 자원이 5개, C 자원이 7개가 있다.



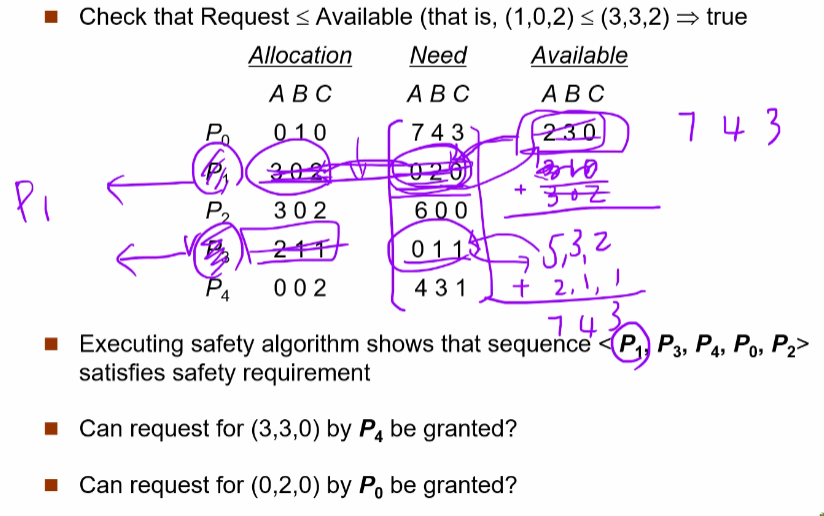
Need 행렬의 값은 (Max – Allocation)으로부터 얻어진다.



이 시스템은 안전하다.

<T1, T3, T4, T2, T0> 순서는 안전성 기준을 만족시키기 때문이다.

T1이 A 자원 한 개와 C 자원 두 개를 추가로 요청한다고 가정해 보자.



<T1, T3, T4, T0, T2>가 안전성 조건을 만족시킨다. 따라서 T1의 요청을 즉시 들어줄 수 있다.

이러한 상태에서 T4가 (3, 3, 0)을 요청하면 자원이 모자라므로 들어줄 수 없다는 것을 알 수 있다.

또 T0가 (0, 2, 0)을 요청한다면 자원은 충분히 있지만 상태를 불안전 상태로 만드므로 역시 그 요청을 즉시 들어줄 수 없다는 것을 알 수 있다.

**8.7 Deadlock Detection; 교착 상태 탐지**

만일 시스템이 교착 상태 예방이나 교착 상태 방지 알고리즘을 사용하지 않는다면, 교착 상태가 발생할 수 있다.

이러한 환경에서는 다음 알고리즘들을 반드시 지원해야 한다.

- 교착 상태가 발생했는지 검사하기 위해 시스템의 상태를 검사하는 알고리즘

- 교착 상태로부터 회복하는 알고리즘

**8.7.1 각 자원 유형이 한 개씩 있는 경우**

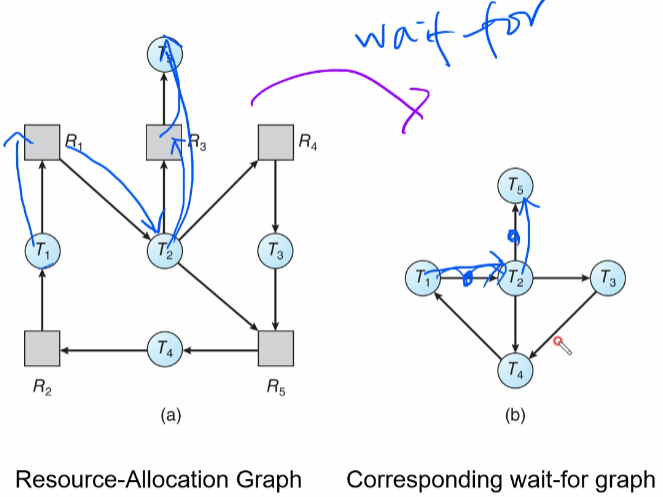
모든 자원이 한 개의 인스턴스만 있다면, 대기 그래프(wait-for graph)라고 하는, 자원 할당 그래프의 변형을 사용해 교착 상태 탐지 알고리즘을 정의할 수 있다.

우리는 자원 할당 그래프로부터 자원 유형의 노드를 제거하고 적절한 간선들을 결합함으로써 대기 그래프를 얻을 수 있다.

앞에서와 마찬가지로, 대기 그래프가 사이클을 포함하는 경우에만 시스템에 교착 상태가 존재한다.

교착 상태를 탐지하기 위해 시스템은 대기 그래프를 유지할 필요가 있고, 주기적으로 그래프에서 사이클을 탐지하는 알고리즘을 호출한다.

(a) 자원 할당 그래프 (b) 대응되는 대기 그래프



**8.7.2 각 유형의 자원을 여러 개 가진 경우**

대기 그래프는 종류마다 자원이 여러 개씩 존재하는 상황에서는 사용할 수 없다.

아래의 기법은 이러한 상황에서 교착 상태를 탐지할 수 있다.

이 알고리즘은 은행원 알고리즘과 마찬가지로 시시각각 그 내용이 달라지는 자료구조를 사용한다.

- **Available**: 각 종류의 자원이 현재 몇 개가 가용한지를 나타내는 벡터

- **Allocation**: 각 스레드에 현재 할당된 자원의 개수를 나타내는 행렬

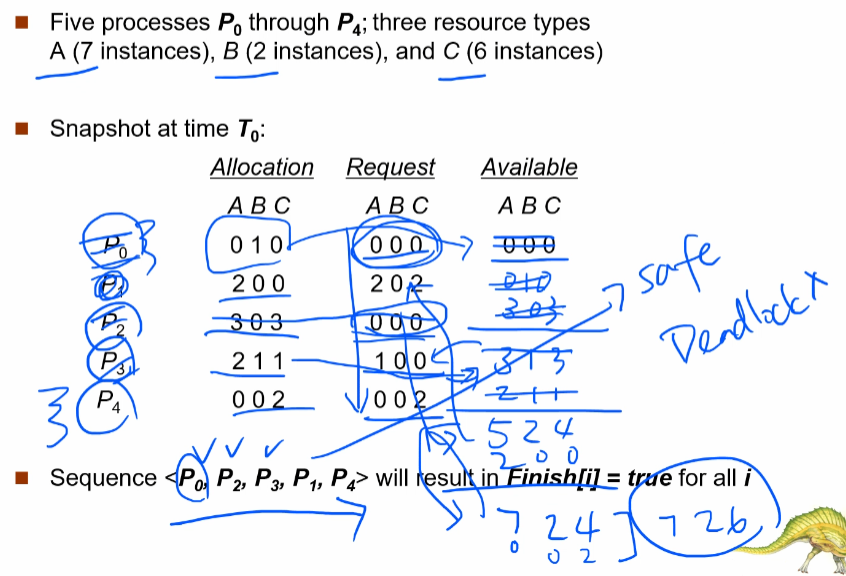
- **Request**: 각 스레드가 현재 요청 중인 자원의 개수를 나타내는 행렬

탐지 알고리즘의 예시

다섯 개의 스레드 T0부터 T4까지 있고

A, B, C 세 가지 종류의 자원이 있다고 가정한다

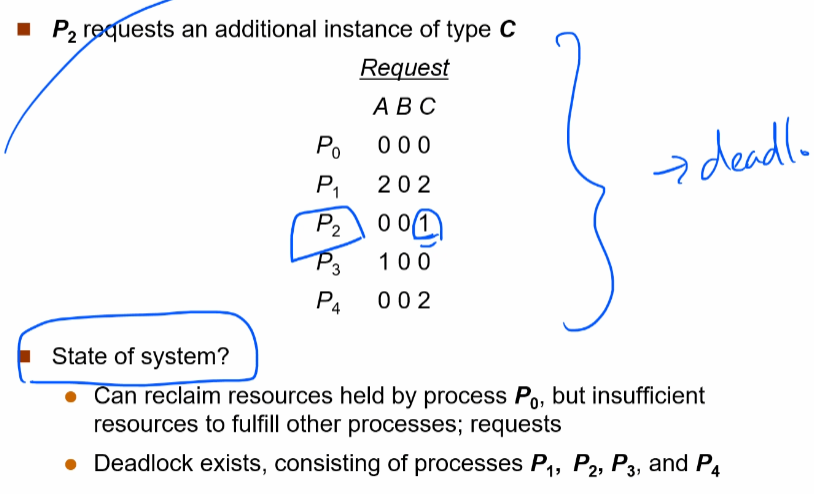
A 자원이 7개, B 자원이 2개, C 자원이 6개가 있다



이 시스템은 교착 상태에 처해있지 않음을 알 수가 있다.

알고리즘을 돌려 보면 <T0, T2, T3, T1, T4> 순서와 같이 작업을 다 끝낼 수 있고 이때 모든 i에 대해서 Finish[i] == true가 됨을 확인할 수가 있다.

그렇지만 T2가 C 자원을 한 개 더 요청하면 Request 행렬은 아래처럼 된다



이제는 시스템이 교착 상태에 빠지게 된다.

T0의 자원을 회수한다고 하더라도 다른 프로세스들이 요구하는 자원을 충족시켜줄 방법이 없기 때문이다.

**8.8 Recover from Deadlock; 교착 상태로부터 회복**

탐지 알고리즘이 교착 상태가 존재한다고 결정하면, 여러 대안의 처리 방법이 있다.

한 가지 방법은 교착 상태가 발생한 것을 운영자(operator)에게 통지해, 운영자가 수작업으로 처리하게 하는 것이다.

다른 방법은 시스템이 자동으로 교착 상태로부터 회복하게 하는 것이다.

교착 상태를 깨뜨리는 데는 두 가지 방법이 있다.

한 가지 방법은 순환 대기를 깨뜨리기 위해 단순히 한 개 이상의 스레드를 중지(abort)시키는 것이다.

두 번째 방법은 교착 상태에 있는 하나 이상의 스레드들로부터 자원을 선점(preempt)하는 것이다.

**8.8.1 Process and Thread Termination; 프로세스와 스레드의 종료**

- **교착 상태 프로세스를 모두 중지**: 이 방식은 확실하게 교착 상태의 사이클을 깨뜨리지만, 그 비용이 크다. 왜냐하면, 이들 프로세스가 오랫동안 연산했을 가능성이 있으며, 이들 부분 계산의 결과들을 반드시 폐기해야 하며, 아마도 후에 다시 계산해야 하기 때문이다.

- **교착 상태가 제거될 때까지 한 프로세스씩 중지**: 이 방법은 각 프로세스가 중지될 때마다 교착 상태 탐지 알고리즘을 호출해 프로세스들이 아직도 교착 상태에 있는지 확인해야 하므로 상당한 오버헤드를 유발한다.

어느 프로세스를 선택할지 결정하는 데는 다음과 같은 많은 결정 요인이 있다.

1. 프로세스의 우선수위가 무엇인지

2. 지금까지 프로세스가 수행된 시간과 지정된 일을 종료하는 데 더 필요한 시간

3. 프로세스가 사용한 자원 유형과 수(예를 들어, 자원들을 선점하기가 단순한지 여부)

4. 프로세스가 종료하기 위해 더 필요한 자원의 수

5. 얼마나 많은 수의 프로세스가 종료되어야 하는지

**챕터9 Main Memory; 메인 메모리**

**9.1 Background; 배경**