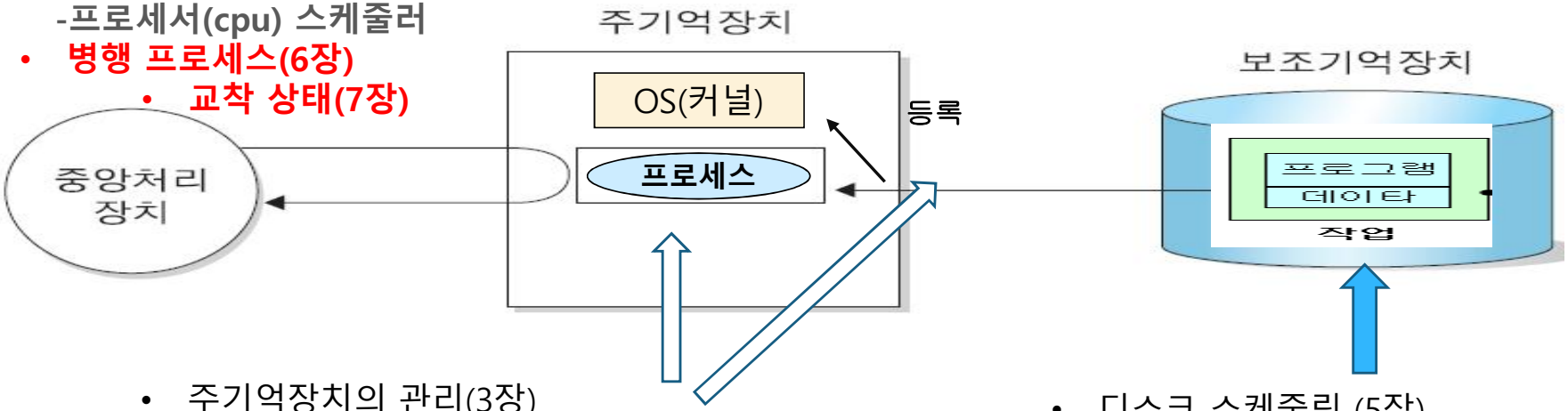
운 영 체 제

7장 교착상태

7장 교착 상태

- **프로세서 관리(2장)**
  -프로세서(cpu) 스케줄러
- **병행 프로세스(6장)**
  - **교착 상태(7장)**



- 주기억장치의 관리(3장)
  -주기억장치 관리 기법
  =>인출 기법,배치 기법,교체 기법, 할당 기법
- 가상 메모리 관리(4장)
  -분산 할당 기법
  =>페이징 기법, 세그먼테이션 기법
  -페이지 교체 기법

- 디스크 스케줄링 (5장)
  -FCFS, SSTF, SCAN, C-SCAN
  => 탐색시간 최소화
- 파일 시스템(5장)
  -파일 구조
  -디렉토리 구조

7장 교착 상태

7장 교착 상태

- 교착 상태의 개요
- 교착 상태 발생 조건
  - 상호배제 (Mutual Exclusion)
  - 점유와 대기(Hold and Wait)
  - 비선점(No preemption)
  - 환형(순환) 대기(Circular Wait)
- 교착 상태의 표현
- 교착 상태의 해결 방법
  - 예방기법(prevention)
  - 회피기법(avoidance)
  - 발견기법(detection)
  - 회복기법(recovery)

- 6장 병행 프로세스
  - 병행 프로세스
  - 병행 처리의 문제점
  - 병행 처리의 문제점 해결 방법
    - 임계 구역
    - 상호 배제 기법
    - 동기화 기법
      - 세마포어
      - 모니터
    - 교착 상태 해결

학습 내용

- 7장 교착 상태
  - 교착 상태의 개요
  - 교착 상태 발생 조건
    - 상호배제 (Mutual Exclusion)
    - 점유와 대기(Hold and Wait)
    - 비선점(No preemption)
    - 환형(순환) 대기(Circular Wait)
  - 교착 상태의 표현

병행 처리

- 병행 프로세스 개념
  - 운영체제가 프로세서를 빠르게 전환, 프로세서 시간 나눠 마치 프로세스 여러 개를 동시에 실행하는 것처럼 보이게 하는 것
- 병행 프로세스의 문제는 컴퓨터 시스템의 자원에는 어느 한 시점에 하나의 프로세스가 할당되어 수행되는데, 동시에 두 개 이상의 프로세스를 병행처리 하면 여러 가지 문제점이 발생

  =>해결방법 : 임계구역, 상호 배제 기법, 동기화 기법(세마포어, 모니터), 교착 상태 해결

상호배제 방법들

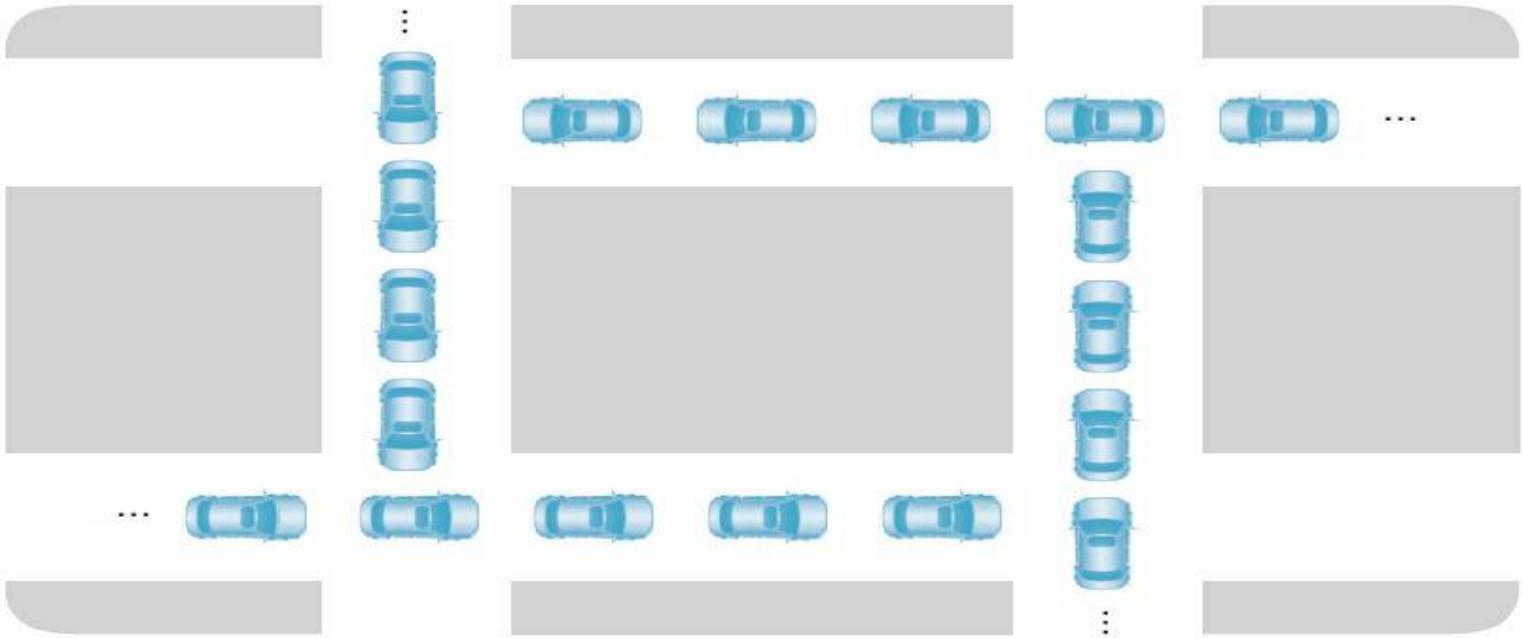
수준	방법	종류
고급	소프트웨어로 해결	• 데커의 알고리즘 • 램포트의 베이커리(빵집) 알고리즘 • 크누스의 알고리즘 • 핸슨의 알고리즘 • 다익스트라의 알고리즘
	소프트웨어가 제공 : 프로그래밍 언어와 운영체제 수준에서 제공	• 세마포 • 모니터
저급	하드웨어로 해결 : 저급 수준의 원자 연산	TestAndSet[TAS](테스)

교착 상태 개요

- 교착 상태(Dead Lock) 개념
  - 컴퓨터 시스템은 한정된 수의 자원(resource)으로 구성됨
  - 교착상태란
    - 하나 또는 그 이상의 프로세스가 발생될 수 없는 어떤 특정 사건(event)을 기다리고 있는 상태
    - 특정 프로세스가 특정한 자원을 위하여 무한정 기다려도 도저히 해결 할 수 없는 상태
  - 교착상태는 컴퓨터 시스템의 효율을 급격히 떨어뜨리는 문제점을 발생시킴

교착 상태 개요

- 일상 생활에서 교착 상태



교착 상태 예 : 교통마비 상태

교착 상태를 해결하기 위한 외부 간섭 필요

교착 상태 개요

- 교착 상태 정의
  - 서로 강의 반대방향으로 향하는 두 사람이 징검다리를 건너면서 같은 돌을 디디려 할 때 발생하는 문제, 두 사람 모두 강을 건널 수 없게 된다. ➔ 교착상태발생
  - **프로토콜(protocol)**을 이용하여 해결
    - 반대편에서 강을 건너는 사람의 유무를 확인하는 것
    - 두 사람이 동시에 출발해도 교착 상태가 발생하고, 두 사람이 모두 기다려도 교착 상태가 발생
    - 하나 이상의 프로세스가 강을 건너기 위해 무작정 기다리는 경우 ➔ 기아상태(starvation) 라고 함

교착 상태 개요

- 정상적인 프로세스의 자원 이용 순서

① 자원 요청(Request) :

– 프로세스가 필요한 자원 요청

– 해당 자원이 다른 프로세스가 사용 중이면 요청을 수락 때까지 대기.

(다른 프로세스에 의하여 자원이 사용 중일 때, 요청한 자원을 얻을 수 있을 때까지 기다려야 한다.)

② 자원 사용(Use) :

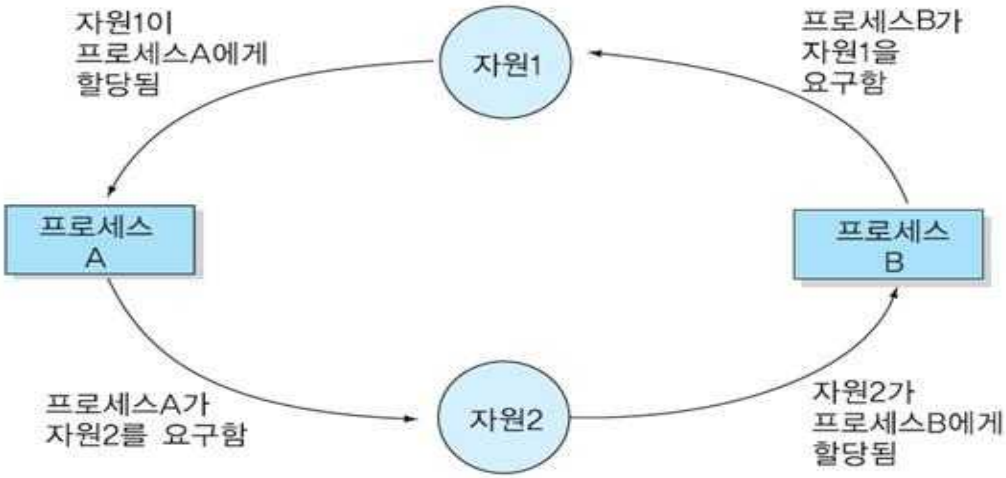
– 프로세스가 요청한 자원 획득하여 사용

③ 자원 해제(Release) :

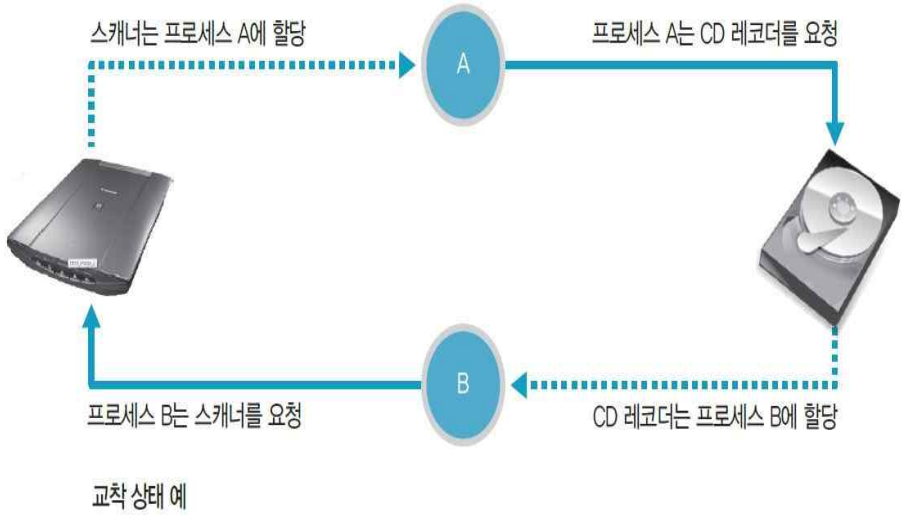
– 프로세스가 자원 사용 마친 후 해당 자원 되돌려(해제) 줌

교착 상태 개요

- 교착상태의 모델
  - 간단한 자원 교착 상태
  - 프로세스가 교착 상태에 빠지면 작업 정지되어 명령 진행 불가
  - 운영체제가 교착 상태 해결 못하면, 시스템 운영자나 사용자는 작업 교체, 종료하는 외부 간섭으로 해결해야 함
  - 하나 이상의 작업에 영향을 주어 무한 대기, 기아 상태보다 더 심각한 문제 야기



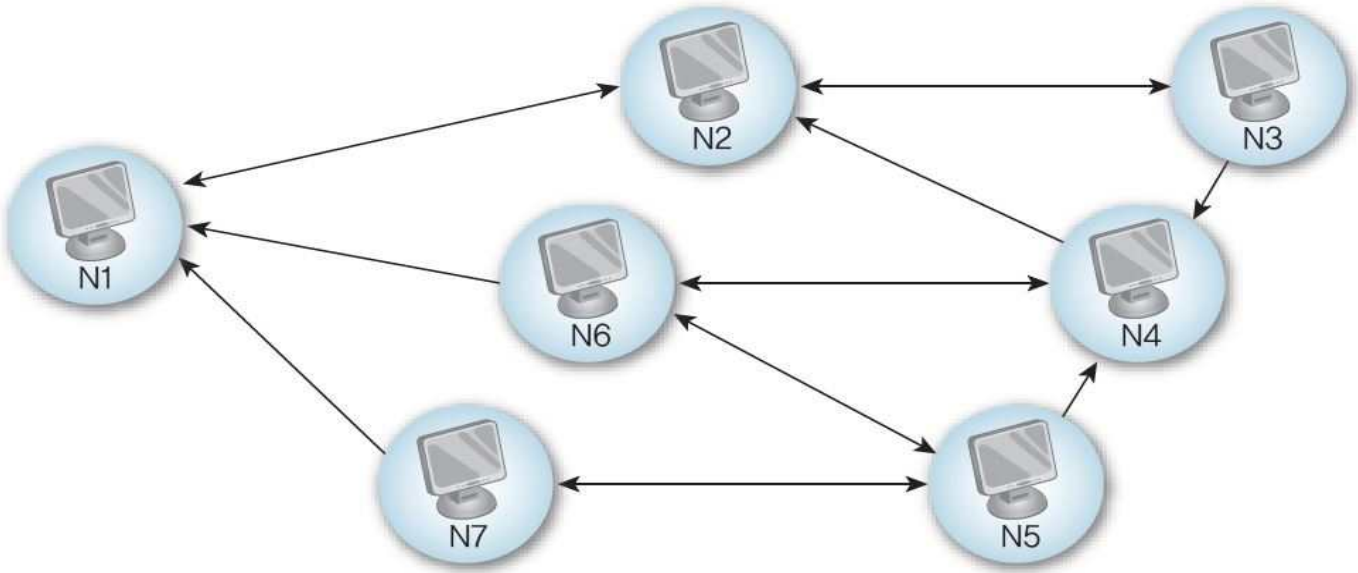
간단한 자원 교착 상태



교착 상태 예

교착 상태 개요

- 네트워크에서 발생하는 교착 상태
- 네트워크가 붐비거나 입출력(I/O) 버퍼 공간이 부족한 네트워크 시스템에 메시지 흐름 제어하는 적절한 프로토콜 없으면, 교착 상태가 발생
- 예



네트워크에서 발생하는 교착 상태

교착 상태의 발생 조건

-교착 상태가 발생하기 위해서는 위의 4가지 조건이 모두 만족되어야 한다.
-환형 대기 조건은 점유와 대기 조건을 포함하므로 4개의 조건이 완전히 독립된 것이 아니지만, 이 조건들을 분리해서 생각하면 유용한 점이 많다.
-이들 조건 중 최소한 한 가지를 일어나지 않게 하여 교착 상태를 예방할 수 있다.

교착 상태 발생의 네 가지 조건

❶ 상호배제(Mutual Exclusion)

- 자원을 최소 하나 이상 비공유. 즉, **한 번에 프로세스 하나만 해당 자원 사용할 수 있어야 함**
- 사용 중인 자원을 다른 프로세스가 사용하려면 요청한 자원 해제될 때 까지 대기

❷ 점유와 대기 (Hold and Wait)

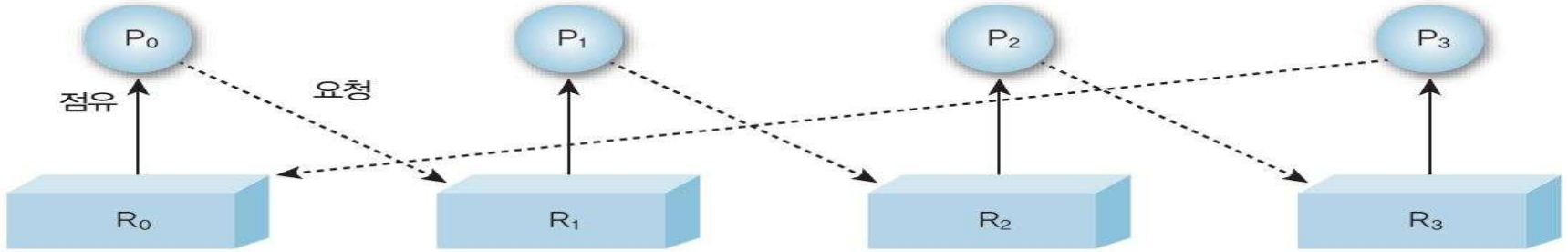
- 자원을 최소한 하나 정도 보유하면서 다른 프로세스에 할당된 자원을 얻으려고 대기하는 프로세스 있어야 함

❸ 비선점 (No preemption)

- 자원 선점 불가. 즉, 다른 프로세스에 할당된 자원은 사용이 끝날 때까지 강제로 빼앗을 수 없어야 함.

❹ 순환(환형) 대기(Circular Wait)

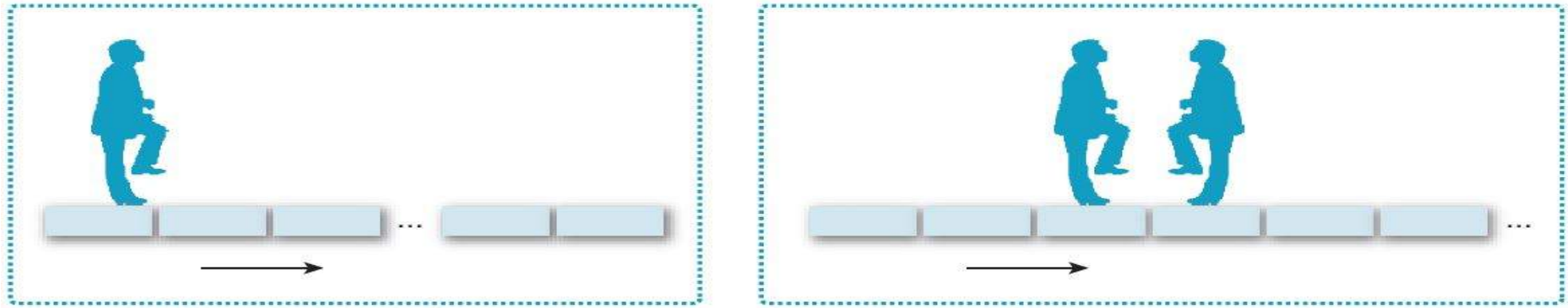
- 공유 자원과 공유 자원을 사용하기 위해 대기하는 프로세스들이 원형으로 구성되어 있어 자신에게 할당된 자원을 점유하면서 앞이나 뒤에 있는 프로세스의 장원 요구해야 함.
- 예



순환 대기의 교착 상태

교착 상태의 발생 조건

교착 상태의 예



강 건너기 예로 살펴본 교착 상태

·상호배제 : 돌 하나를 한 사람만 디딜 수 있음
·점유와 대기 : 각 사람은 돌 하나를 딛고 다음 돌을 요구
·비선점 : 사람이 딛고 있는 돌을 강제로 제거할 수 없음
·순환 대기 : 왼쪽에서 오는 사람은 오른쪽에서 오는 사람 기다리고, 오른쪽에서 오는 사람도 왼쪽에서 오는 사람 기다림

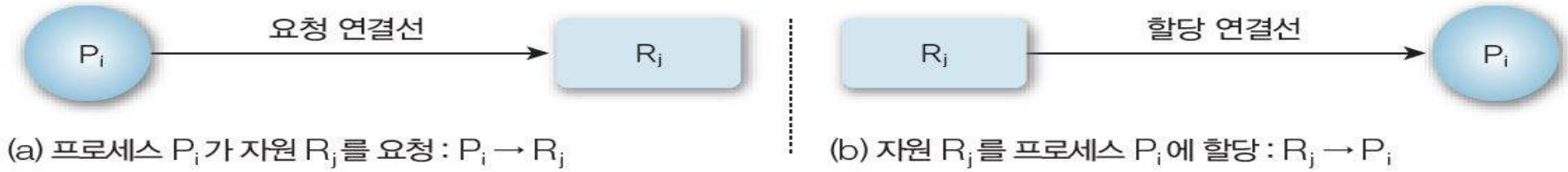
- 교착 상태 해결 방법
  - ❶ 둘 중 한 사람이 되돌아간다(복귀).
  - ❷ 징검다리 반대편을 먼저 확인하고 출발한다.
  - ❸ 강의 한편에 우선순위를 부여

교착 상태의 표현

교착 상태의 표현

- 시스템 자원 할당 그래프인 방향 그래프 표현

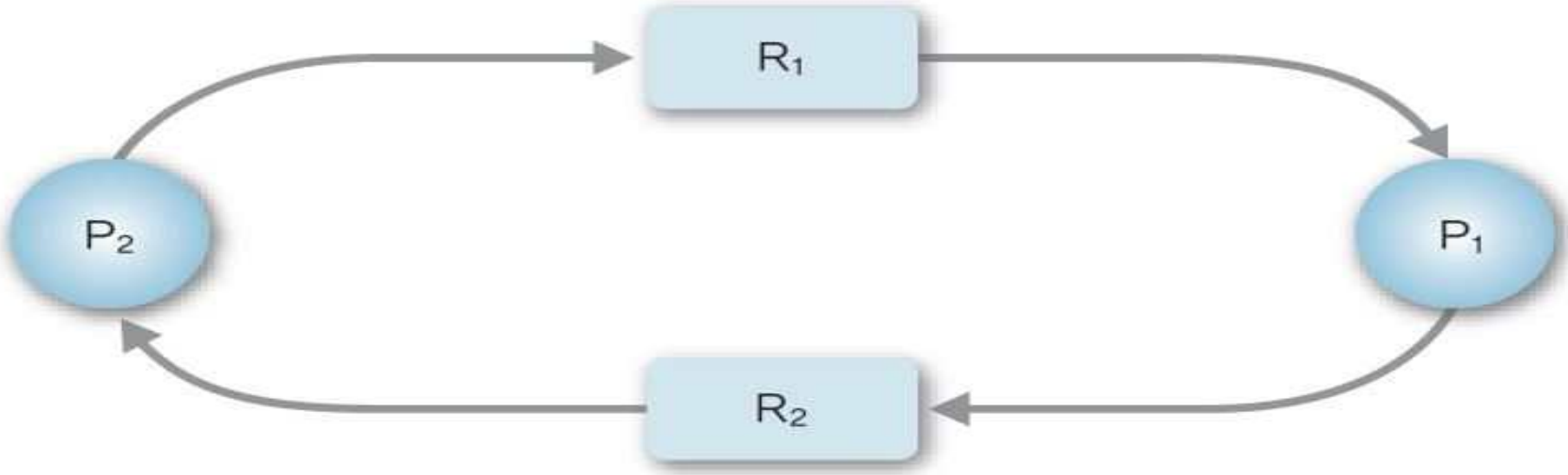
- 자원 할당 그래프
  - G= (V, E)로 구성, 정점 집합 V는 프로세스 집합 P= {P1, P2, ……, Pn}과 자원 집합 R= {R1, R2, ……, Rn}으로 나뉨. 간선 집합 E는 원소를 (Pi, Rj)나 (Rj, Pi)와 같은 순서쌍으로 나타냄
  - 예



자원 할당 그래프

교착 상태의 표현

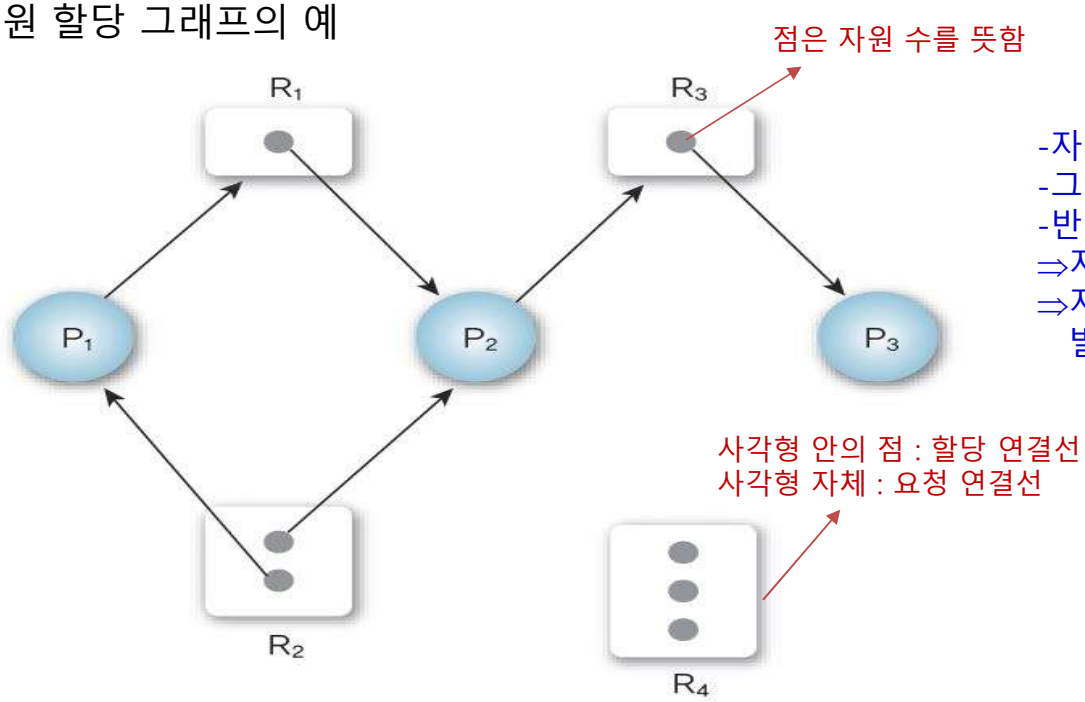
- 사이클이 있어 교착 상태인 자원 할당 그래프



사이클이 있어 교착 상태인 자원 할당 그래프 : 프로세스 P_1, P_2가 교착 상태

교착 상태의 표현

- 자원 할당 그래프의 예



-자원 할당 그래프를 통해
-그래프에 사이클이 없다면 시스템에 교착상태는 발생하지 않는다.
-반면에 사이클이 있으면 교착 상태임을 암시한다.
⇒자원 하나에 사이클이 연관 되어 있으면 교착 상태가 꼭 발생
⇒자원이 여러 개 일 경우는 사이클이 있다고 반드시 교착 상태가 발생한다는 의미는 아니다.

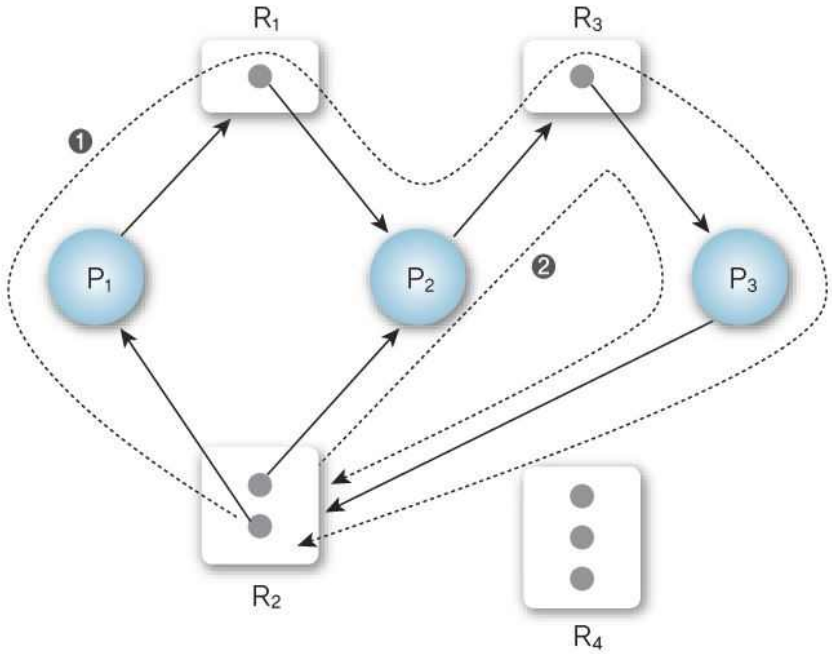
❶ 집합 P, R, E
- $P = \{P_1, P_2, P_3\}$
- $R = \{R_1, R_2, R_3, R_4\}$
- $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$

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.

자원 할당 그래프 예와 프로세스 상태

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

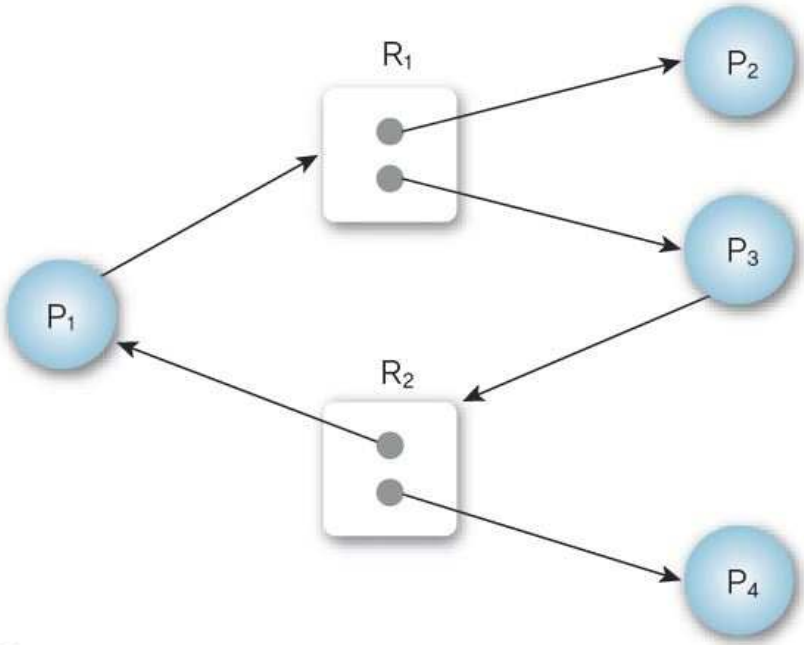
교착 상태의 할당 그래프와 사이클

-자원 할당 그래프를 통해
-그래프에 사이클이 없다면 시스템에 교착상태는 발생하지 않는다.
-반면에 사이클이 있으면 교착 상태임을 암시한다.
⇒자원 하나에 사이클이 연관 되어 있으면 교착 상태가 꼭 발생
⇒자원이 여러 개 일 경우는 사이클이 있다고 반드시 교착 상태가 발생한다는 의미는 아니다.

프로세스 P3이 자원 R2의 자원을 요청한다고 가정할 때
그래프에 있는 사이클이 2개가 있고 P1, P2, P3은 교착 상태이다.
=>프로세스 P2는 프로세스 P3이 점유한 자원 R3을 기다리고, 프로세스P3은 자원 R2를 얻으려고
프로세스 P1이나 프로세스 P2가 자원 R2를 해제할 때까지 기다리기 때문에 교착 상태가 발생

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

사이클이 있으나 교착 상태가 아닌 할당 그래프

-자원 할당 그래프를 통해
-그래프에 사이클이 없다면 시스템에 교착상태는 발생하지 않는다.
-반면에 사이클이 있으면 교착 상태임을 암시한다.
⇒자원 하나에 사이클이 연관 되어 있으면 교착 상태가 꼭 발생
⇒자원이 여러 개 일 경우는 사이클이 있다고 반드시 교착 상태가 발생한다는 의미는 아니다.(시스템은 교착 상태일 수도 있고 아닐 수도 있음)

⇒**이러한 관찰은 교착 상태 문제 해결 방법에 중요한 요건이 된다.**

사이클이 있지만 교착 상태가 없다,
=> 프로세스 P4가 자원 R2를 해제할 수 있고, 해제한 자원을 P3에 할당하면 사이클이 없어지기 때문

학습 내용

- 7장 교착 상태
  - 교착 상태의 해결 방법
    - 예방기법(prevention)
    - 회피기법(avoidance)
    - 발견기법(detection)
    - 회복기법(recovery)

교착상태 해결기법

- 교착상태 해결 기법
  - 교착상태 예방(prevention) 기법
  - 교착상태 회피(avoidance) 기법
  - 교착상태 발견(detection) 기법
  - 교착상태 회복(recovery) 기법

예방기법(prevention)

- 예방 기법(Prevention) : 교착 상태가 발생되지 않도록 사전에 시스템을 제어하는 방법으로, 교착 상태 발생의 4가지 조건 중에서 어느 하나를 제거(부정)함으로써 수행됨

상호 배제 (Mutual Exclusion) 부정	한 번에 여러 개의 프로세스가 공유 자원을 사용할 수 있도록 하는 것으로 실현은 불가능함
점유 및 대기 (Hold and Wait) 부정	프로세스가 실행되기 전 필요한 모든 자원을 할당하여 프로세스 대기를 없애거나 자원이 점유되지 않은 상태에서만 자원을 요구하도록 함
비선점(Non-preemption) 부정	자원을 점유하고 있는 프로세스가 다른 자원을 요구할 때 점유하고 있는 자원을 반납하고, 요구한 자원을 사용하기 위해 기다리게 함
환형 대기 (Circular Wait) 부정	자원을 선형 순서로 분류하여 고유 번호를 할당하고, 각 프로세스는 현재 점유한 자원의 고유 번호보다 앞이나 뒤 어느 한쪽 방향으로만 자원을 요구하도록 함

- ❖ **자원의 낭비가 가장 심한 기법이다.**
- ❖ **예방기법 중 점유 및 대기 부정은 각 프로세스는 한꺼번에 자기에게 필요한 자원을 모두 요구해야 하며, 이 요구가 만족되지 않으면 작업을 진행할 수 없게 하는 방법**
- ❖ **현실적으로 사용하기 어려움**

예방기법(prevention)

- **하벤더(Havender)의 교착 상태 예방 방법**
  - 하벤더(Havender)는 네가지 조건 중 **상호배제조건**을 제외한 세가지 조건의 방지법만 제시 하였음
  - 상호 배제 조건을 부정하지는 않음
    - 이 조건을 부정하면 하나의 자원을 여러 프로세스가 공유함을 허락 하는 꼴
  - 점유와 대기 조건 방지
    - 각 프로세스는 필요한 자원 한 번에 모두 요청해야 하며, 요청한 자원을 모두 제공받기 전 까지는 작업 진행 불가
  - 비선점 조건 방지
    - 어떤 자원을 점유하고 있는 프로세스의 요청을 더 이상 허용하지 않으면 점유한 자원을 모두 반납하고, 필요할 때 다시 자원 요청
  - 순환(환형) 대기 조건 방지
    - 모든 프로세스에 자원 순서대로 할당

예방기법(prevention)

- **상호 배제 조건 방지**
  - 시스템 내의 모든 자원들을 공유 가능하게 하는 방법
  - 불가능함
- **점유와 대기조건 방지**
  - 각 프로세스는 필요한 자원들을 모두 한꺼번에 신청
    요구한 자원을 전부 할당하여 주거나, 또는 하나라도 부족하면 전혀 할당하지 않는 방법
  - 자원의 낭비를 초래할 수 있어 비용이 증가될 수 있다
  - 한꺼번에 원하는 자원을 제공할 수 없는 경우에는 무한 연기의 원인이 될 수 있다
  - 추가 자원에 대한 요구가 거절당했다면 그 프로세스는 자원을 전부 반납
- **비선점 조건 방지**
  - 어떤 자원을 점유하고 있는 프로세스의 요청을 더 이상 허용하지 않으면 점유한 자원을 모두 반납하고, 필요할 때 다시 자원 요청
  - 자원의 낭비가 매우 심함
  - 비실현적임
- **환형 대기조건 방지**
  - 모든 자원에게 고유 번호를 부여
  - 현재 가지고 있는 자원보다 더 큰 번호를 가진 자원만을 요청하도록 제한하는 방법
  - 문제점
    - 예상한 순서와 다르게 자원을 요구하는 작업들은 실제로 자원을 사용하기 훨씬 전부터 자원을 요구·점유하고 있어야 하므로 비용증가와 성능감소가 있을 수 있다.
    - 새로운 자원이 시스템에 추가되면 현존하는 프로그램과 시스템을 재 작성 해야 함

회피기법(Avoidance)

- 회피 기법(Avoidance) : 교착 상태가 발생할 가능성을 배제하지 않고, 교착 상태가 발생하면 적절히 피해나가는 방법으로, 주로 은행원 알고리즘(Banker's Algorithm)이 사용됨

은행원 알고리즘	• Dijkstra가 제안한 것으로, 은행에서 모든 고객의 요구가 충족되도록 현금을 할당하는 데서 유래한 기법 • 각 프로세스에게 자원을 할당하여 교착 상태가 발생하지 않으며 모든 프로세스가 완료될 수 있는 상태를 안전 상태, 교착 상태가 발생할 수 있는 상태를 불안전 상태라고 함

- ❖ 은행원 알고리즘을 적용하기 위해서는 자원의 양과 사용자(프로세스) 수가 일정 해야 한다.
- ❖ 은행원 알고리즘은 프로세스의 모든 요구를 유한한 시간 안에 할당하는 것을 보장한다.
- ❖ 은행원 알고리즘은 대화식 시스템에 적용할 수 없다.

회피기법(Avoidance)

교착 상태 회비의 개념

- 목적 : 덜 엄격한 조건 요구하여 자원 좀 더 효율적 사용

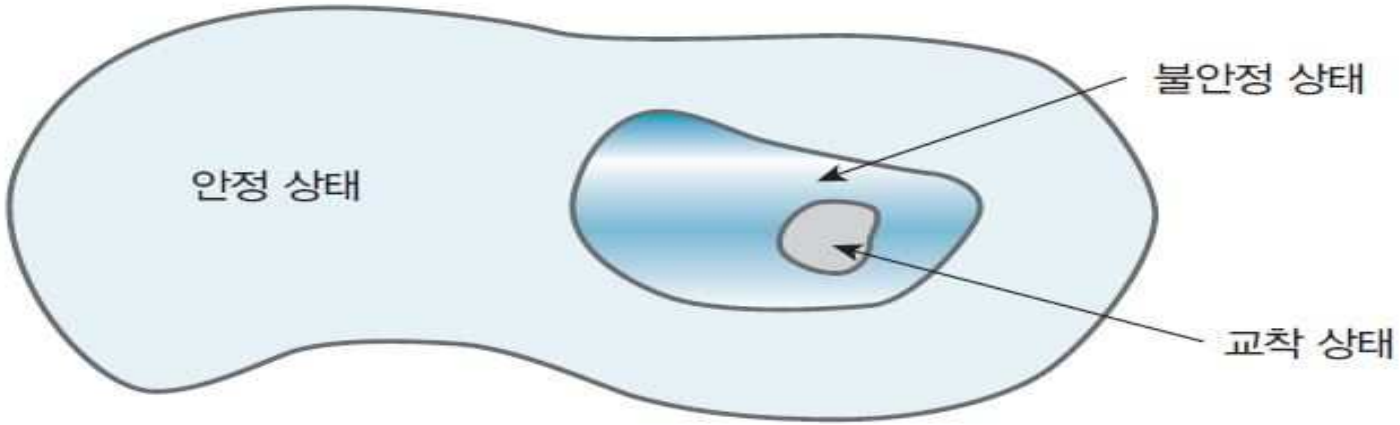
- 교착 상태의 모든 발생 가능성을 미리 제거하는 것이 아닌 교착 상태 발생할 가능성 인정하고(세 가지 필요조건 허용), 교착 상태가 발생하려고 할 때 적절히 회피하는 것

- 예방보다는 회피가 더 병행성 허용

- 교착 상태의 회피 방법
  - 프로세스의 시작 중단
    - 프로세스의 요구가 교착 상태 발생시킬 수 있다면 프로세스 시작 중단
  - 자원 할당 거부(은행원 알고리즘)
    - 프로세스가 요청한 자원 할당했을 때 교착 상태 발생할 수 있다면 요청한 자원 할당 않음

회피기법(Avoidance)

- **시스템의 상태**
  - 안정 상태와 불안정 상태로 구분
  - 교착 상태는 불안정 상태에서 발생
  - 모든 사용자가 일정 기간 안에 작업을 끝낼 수 있도록 운영체제가 할 수 있으면 현재 시스템의 상태가 안정, 그렇지 않으면 불안정
  - 교착 상태는 불안정 상태. 그러나 모든 불안정 상태가 교착 상태인 것은 아님, 단지 불안정 상태는 교착 상태가 되기 쉬움.
  - 상태가 안정하다면 운영체제는 불안정 상태와 교착 상태를 예방 가능
  - 불안정 상태의 운영체제는 교착 상태 발생시키는 프로세스의 자원 요청 방지 불가



안정 상태와 불안정 상태, 교착 상태의 공간

회피기법(Avoidance)

- **Dijkstra's** 알고리즘(다익스트라의 은행가 알고리즘)
  - 교착상태 회피를 위한 간단한 이론적 기법
  - 자원 할당 거부, 시스템의 상태를 항상 안전상태로만 진행 시킴
  - 자원의 할당 허용 여부 결정 전에 미리 결정된 모든 자원의 최대 가능한 할당량을 시뮬레이션하여 안전 여부 검사. 그런 다음 대기 중인 다른 모든 활동의 교착 상태 가능성 조사하여 '안전 상태' 여부 검사·확인
  - 프로세스가 요청한 자원 할당했을 때 교착 상태 발생할 수 있다면 요청한 자원 할당 않음

- 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.

$$Available = Available - Request_i;$$
$$Allocation_i = Allocation_i + Request_i;$$
$$Need_i = Need_i - Request_i;$$

은행가 알고리즘

자원 할당이 안정 상태라면 처리가 되고 있는 프로세스 Pi는 자원 할당받음
불안정 상태라면 Pi는 Requesti를 대기하고 이전 자원 할당의 상태로 복귀

회피기법(Avoidance)

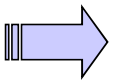
- **Dijkstra's** 알고리즘

안전 상태의 예

◆ 시스템의 가정
- 한가지 종류의 자원 R, 단위 자원 10개, 프로세스 3개

상태 - 1

프로세스-ID	최대 요구량	현재 할당량	추가요구 가능량
P1	3	1	2
P2	9	5	4
P3	5	2	3



◆Available resource units : 2
◆실행 종료 순서 : P1 → P3 → P2
- 안전 순서 (safe sequence)
◆ 현재 상태에서 안전 순서가 하나이상 존재하면 안전 상태임

최대 요구량(Max)
현재 할당량(Allocation)
추가 요구 가능량(Need)=Max-Allocation
잔여량(Available)

회피기법(Avoidance)

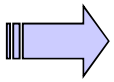
- **Dijkstra's** 알고리즘

비안전 상태의 예

◆ 시스템의 가정
- 한가지 종류의 자원 R, 단위 자원 10개, 프로세스 3개

상태 - 2

프로세스-ID	최대 요구량	현재 할당량	추가요구 가능량
P1	5	1	4
P2	9	5	4
P3	7	2	5



◆Available resource units : 3
◆안전 순서 : 없음
◆임의의 순간에 세 프로세스들이 모두 세 개 이상의 단위 자원을 요청하는 경우 시스템은 교착상태 놓이게 됨

최대 요구량(Max)
현재 할당량(Allocation)
추가 요구 가능량(Need)=Max-Allocation
잔여량(Available)

발견기법(Detection)

- 교착상태 발견기법은 시스템에 교착상태가 발생했는지 점검하여 교착상태에 있는 프로세스와 자원을 발견하는 것을 의미
- 교착상태에 대비한 사전 처리 없음
- 시스템 운영 중 주기적으로 또는 필요한 경우에 교착상태 확인
  - 현재 시스템이 교착상태인지의 여부
  - 현재 시스템이 교착상태라면 교착상태의 프로세스 검출
- 교착상태 발견 알고리즘과 자원 할당 그래프(resource allocation graph) 등을 사용할 수 있음
- 교착상태 발견 알고리즘
  - 교착 상태 발생 빈도수와 교착 상태가 발생했을 때 영향을 받는 프로세스 수에 따라 결정
  - 교착 상태가 자주 발생하면 발견 알고리즘도 더 자주 호출
  - 일단 교착 상태가 발생하면 해결할 때까지 프로세스에 할당된 자원들은 유휴 상태, 교착 상태의 프로세스 수는 점점 증가

회복기법(recovery)

- 교착상태 회복기법은 교착상태를 일으킨 프로세스를 종료하거나 교착상태의 프로세스에 할당된 자원을 선점하여 프로세스나 자원을 회복하는 것을 의미한다.
- 시스템 내에 존재하는 교착상태를 제거

- 회복기법(recovery) 방법
  - 프로세스 종료 (process termination) 기법
    - 교착상태에 있는 프로세스를 종료하는 것으로, 교착상태에 있는 모든 프로세스를 종료하는 방법과 교착상태에 있는 프로세스들을 하나씩 종료해 가며 교착상태를 해결하는 방법이 있다.
  - 자원선점 (resource preemption) 기법
    - 교착상태의 프로세스가 점유하고 있는 자원을 선점하여 다른 프로세스에게 할당하며, 해당 프로세스를 일시 정지시키거나 방법이다.
    - 우선순위가 낮은 프로세스, 수행된 정도가 적은 프로세스 , 사용되는 자원이 적은 프로세스등을 위주로 해당 프로세스의 자원을 선점한다.

회복기법(recovery)

- 프로세스 종료 (process termination) 기법
  - 교착상태의 프로세스들 중 일부를 강제로 종료시킴
  - 종료 비용 (termination cost) 모델 사용
    - 교착상태를 제거하기 위해 종료될 프로세스들을 선택
    - 종료 비용 산출 요소
      - 프로세스 우선순위 (priority)
      - 프로세스의 종류 (type)
      - 각 프로세스의 현재까지의 실행시간
      - 각 프로세스가 실행을 마치기 위하여 앞으로 남은 시간
      - 회계 비용 (accounting cost)
    - 각 프로세스 별로 종료 비용 산출

회복기법(recovery)

- 자원 선점 (resource preemption) 기법
  - 교착상태를 제거하기 위해 선점되어야 할 자원들을 선정
  - 선정된 자원을 선점 당한 프로세스들은 종료되고 이후 다시 실행을 시작함
    - 교착상태에 놓이지 않은 프로세스들도 강제 종료의 대상이 됨
  - 선점 대상 자원들의 선정
    - 자원들의 선점 비용 모델 설정
    - 최소의 비용을 갖는 복구 기법 사용