교착상태

7th Week

Kim, Eui-Jik

Contents

- 소개
- 교착 상태의 예
- 자원의 개념
- 교착 상태가 성립되기 위한 네 가지 필요 조건
- 교착 상태 해결책
- 교착 상태 방지
- 다익스트라의 은행원 알고리즘을 사용한 교착 상태 회피
- 교착 상태 방지
- 교착 상태 복구

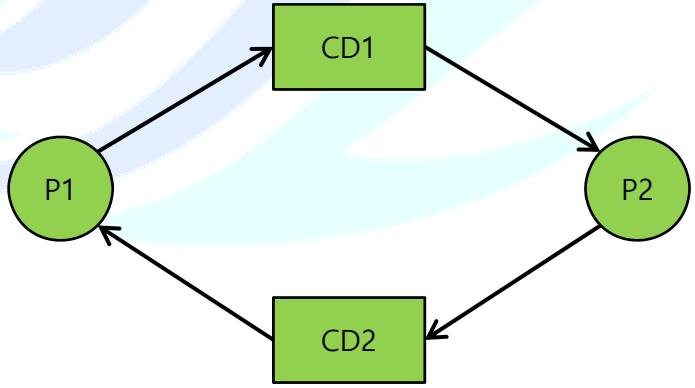
소개

- 교착 상태(Deadlock)
  - 프로세스나 스레드가 결코 일어날 수 없는 특정 이벤트를 기다리는 것
  - 시스템 측면에서 자원의 요구가 뒤엉킨 상태
    - 한 프로세스 집합 내의 프로세스들에 의해 발생할 사건(Event)을 프로세스들이 서로 기다리고 있는 상태
    - 둘 이상의 작업이 보류 상태에 놓여 중요한 자원을 이용하기 위해 기다릴 때 발생함

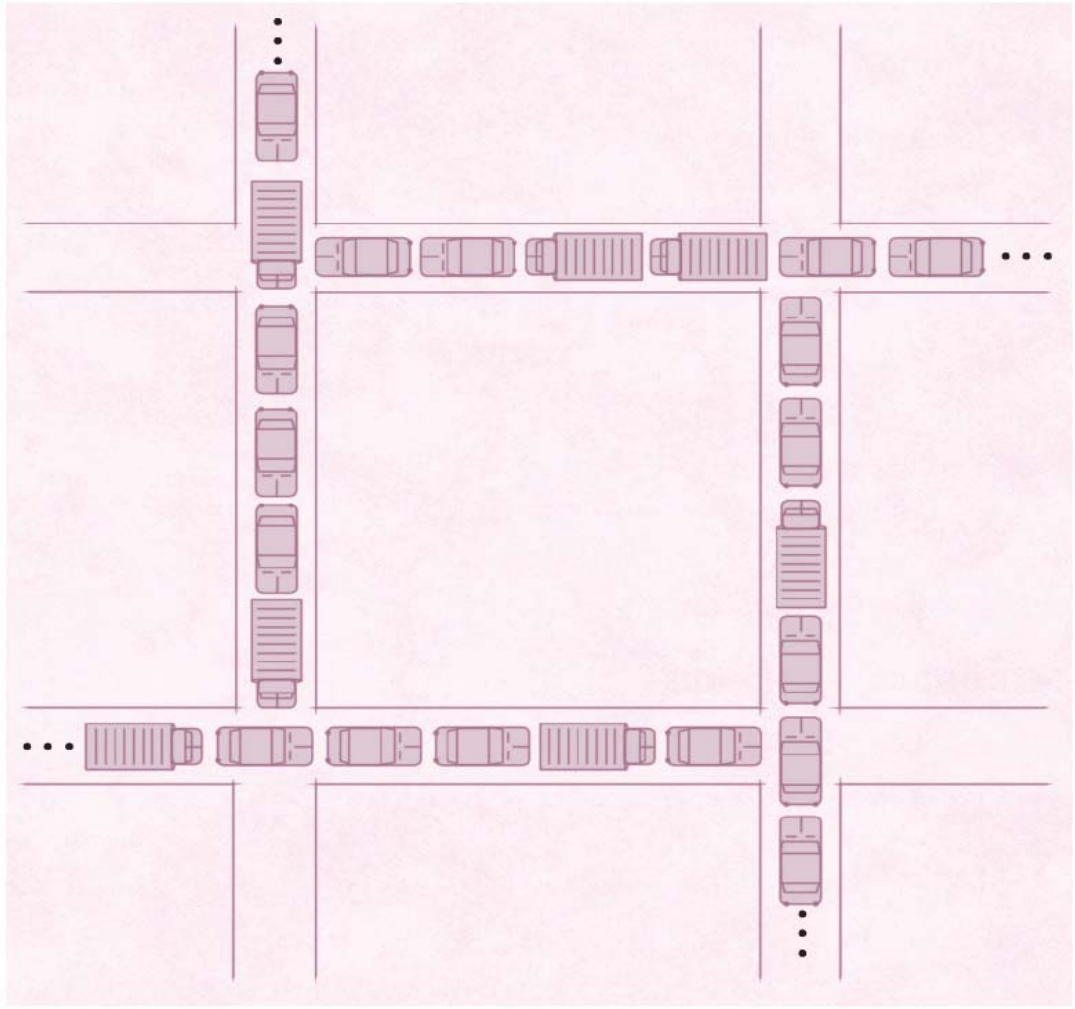
- 시스템 교착 상태
  - 하나 또는 그 이상의 프로세스가 교착 상태에 있는 것

[참고] Deadlock Problem

- 교착상태(Deadlock)
  - 블록된 프로세스 집합의 각 프로세스가 하나의 자원을 소유(holding)하면서 그 집합에 있는 다른 프로세스가 소유하고 있는 자원의 획득을 기다리고 있는 상태
- Example
  - 2개의 CD RW drives 가진 시스템에서
  - P1과 P2가 각기 하나의 CD RW drive를 소유하고 있으면서, 나머지 CD RW drive를 소유하기를 원함



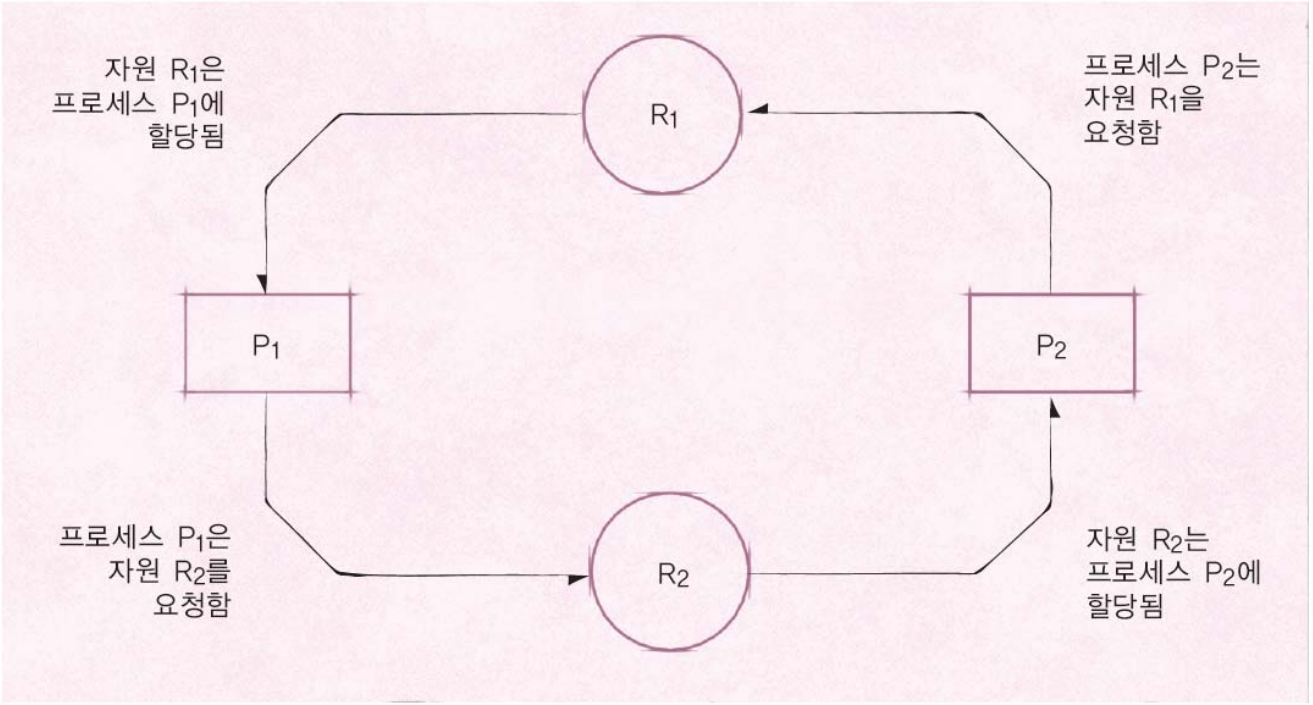
교착 상태의 예



[그림 7-1] 도로에서 발생하는 교착 상태의 예[4]

교착 상태의 예

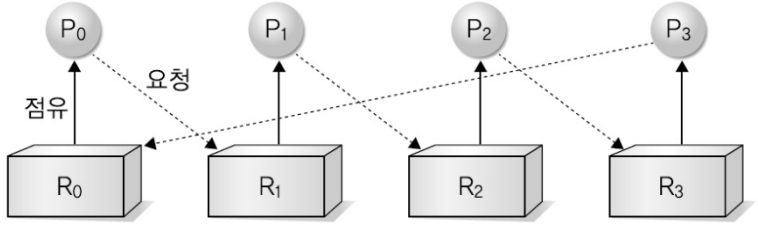
- 간단한 자원 교착 상태
  - 대부분의 교착 상태는 전용 자원을 차지하려는 경쟁에서 발생
  - "순환 대기"는 교착 상태에 있는 시스템의 특징



[그림 7-2] 자원 교착 상태의 예(각 프로세스가 보유한 자원을 다른 프로세스에서 요구하는데, 어떤 프로세스도 자신이 보유한 자원을 해제하려고 하지 않아 교착 상태에 빠진 시스템)

교착 상태가 성립되기 위한 네 가지 필요조건

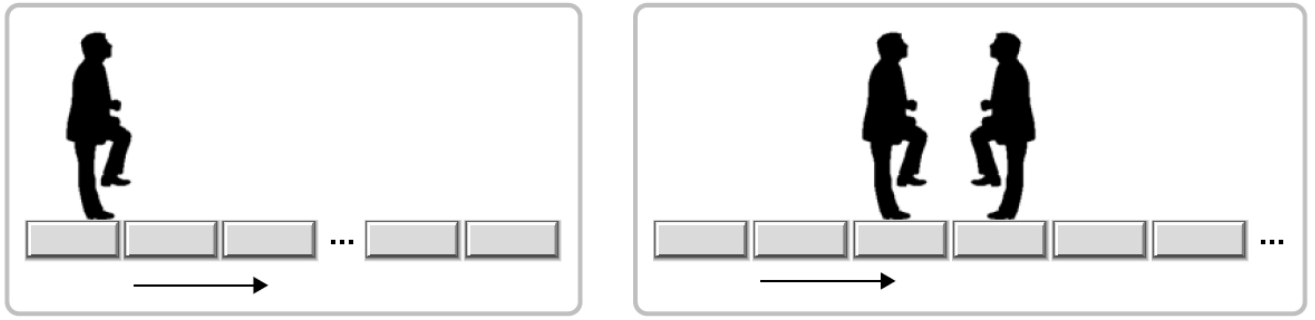
- 다음 네 가지 조건이 동시에 발생할 때, 즉 필요충분조건이 성립될 때 발생
- 상호 배제 조건
  - 자원은 한 번에 한 프로세스에서만 배타적으로 점유
    - 자원이 최소 하나 이상 비공유, 즉 한 번에 한 프로세스만 해당 자원을 사용할 수 있어야 함
    - 사용 중인 자원을 다른 프로세스가 사용하려면 요청한 자원이 해제될 때까지 대기해야 함
- 점유와 대기 조건(보유 후 대기 조건)
  - 배타적으로 자원을 획득한 프로세스가 다른 자원을 얻으려고 대기
    - 최소한 자원 하나를 보유하고 다른 프로세스에 할당된 자원을 얻기 위해 기다리는 프로세스가 있어야 함
- 비선점 조건
  - 프로세스가 자원을 확보하면, 자원을 모두 사용할 때까지 시스템이 프로세스의 제어를 빼앗을 수 없음
    - 자원은 선점될 수 없음. 즉 자원을 강제로 뺏을 수 없으며 자원을 점유하고 있는 프로세스가 끝나야 해제됨
- 순환대기 조건
  - 두 프로세스 이상이 순환고리 형태
  - 대기 프로세스 집합 $\{P_0, P_1, \ldots, P_n\}$이 있을 때, P_0은 P_1이 보유하고 있는 자원을, P_1 은 P_2, P_2 는 P_3, P_{n-1}은 P_n, P_n 은 P_0 이 보유하고 있는 자원을 각각 얻기 위해 대기하는 경우



순환대기의 교착상태

[참고] 강 건너기 교착상태 예

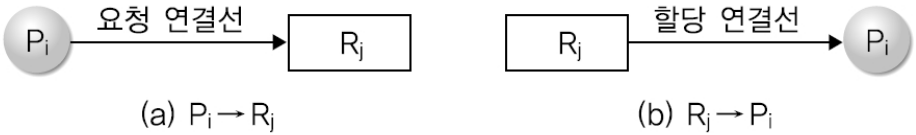
- 강 건너기 교착상태 예
  - 여러 개의 돌로 된 징검다리가 있는 강을 건너는 경우
  - 징검다리의 돌 하나는 한 쪽에서 한 사람만 디딜 수 있음 (상호배제 성립)
  - 강을 건너는 사람을 프로세스, 징검다리의 돌을 자원이라 가정함
    - 두 사람이 동시에 서로 다른 방향에서 출발, 강 중간에서 만나면 교착상태가 발생했다 할 수 있음
    - 돌을 딛는 것을 자원 할당, 발을 떼는 것을 자원 해제로 볼 때 동시에 같은 돌을 디디려 하면 교착상태가 발생함
    - 각 사람은 돌 하나를 딛고 다음 돌을 요구함 (점유와 대기 조건 만족)
    - 사람이 딛고 있는 돌을 강제로 제거할 수 없음 (비선점 조건 만족)
    - 왼쪽에서 오는 사람은 오른쪽 사람을, 오른쪽에서 오는 사람은 왼쪽 사람을 기다림 (순환대기 조건 만족)
  - 해결 방법
    - 둘 중 한 사람이 되돌아간다 (복귀)
    - 강을 건너기 전에 상대편 강 쪽을 확인하고 출발한다
    - 강의 한쪽 편에 우선권을 부여한다



강 건너기 교착상태

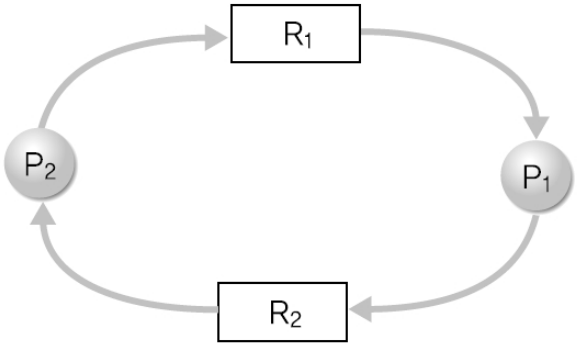
[참고] 자원 할당 그래프

- 자원 할당 그래프를 이용한 교착상태 표현
  - 방향 그래프인 시스템 자원 할당 그래프를 이용하여 교착 상태 기술
    - **G = (V, E)**로 구성, 정점 집합(**V**)은 두 가지 형태로 나뉨
      - 프로세스 집합인 $\mathbf{P} = \{P_1, P_2, \ldots, P_n\}$
      - 시스템 내의 모든 자원들로 구성된 간선 집합(**E**)인 $\mathbf{R} = \{R_1, R_2, \ldots, R_n\}$
    - 집합 **E**의 각 원소는 (P_i, R_j)나 (R_j, P_i) 같은 순서쌍으로 나타내고, P_i로 프로세스를, R_j로 자원을 표시함
      - $P_i \rightarrow R_j$ (요청 연결선) : 프로세스 P_i에서 자원 형태 R_j로의 연결선, 프로세스 P_i가 자원 형태 R_j를 요청하는 상태를 의미함(대기)
      - $R_j \rightarrow P_i$ (할당 연결선) : 자원 형태 R_j에서 프로세스 P_i로의 연결선, 자원 형태 R_j가 프로세스 P_i에 할당됨을 의미함(할당)



(a) $P_i \rightarrow R_j$ (b) $R_j \rightarrow P_i$

$P_i \rightarrow R_j$ 와 $R_j \rightarrow P_i$ 그래프



P_1, P_2 가 교착상태

[참고] 자원 할당 그래프

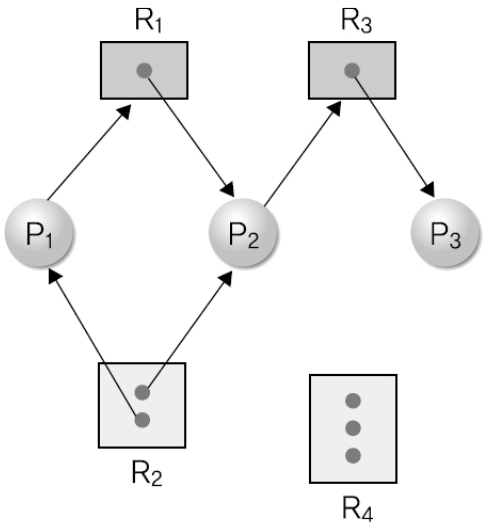
- 자원 할당 그래프 표현
  - 프로세스 P_i는 원, 자원 형태 R_j는 사각형으로 표기함.
  - 원 안에 그려진 작은 원(검은색 점)은 각 집합체의 자원 개수를 나타냄.
  - 할당 연결선은 사각형 안의 작은 점 하나를 가리키고, 요청 연결선은 사각형 R_j만을 가리킴.
    - 프로세스 P_i가 자원 형태 R_j의 한 자원을 요청하면, 요청 연결선을 자원 할당 그래프 안에 삽입함.
    - 요청이 만족 시 요청 연결선은 즉시 할당 연결선으로 변환, 프로세스가 자원을 해제하면 할당 연결선은 삭제됨.
  - [그림]의 자원 할당 그래프는 다음 상황을 의미함.
    - 집합 P, R, E

      $P = \{P_1, P_2, P_3\}$

      $R = \{R_1, R_2, R_3, R_4\}$

      $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$
    - 자원의 사례

      * 자원 형태 R_1은 한 개다.
      * 자원 형태 R_2는 두 개다.
      * 자원 형태 R_3는 한 개다.
      * 자원 형태 R_4는 세 개다.



자원 할당 그래프

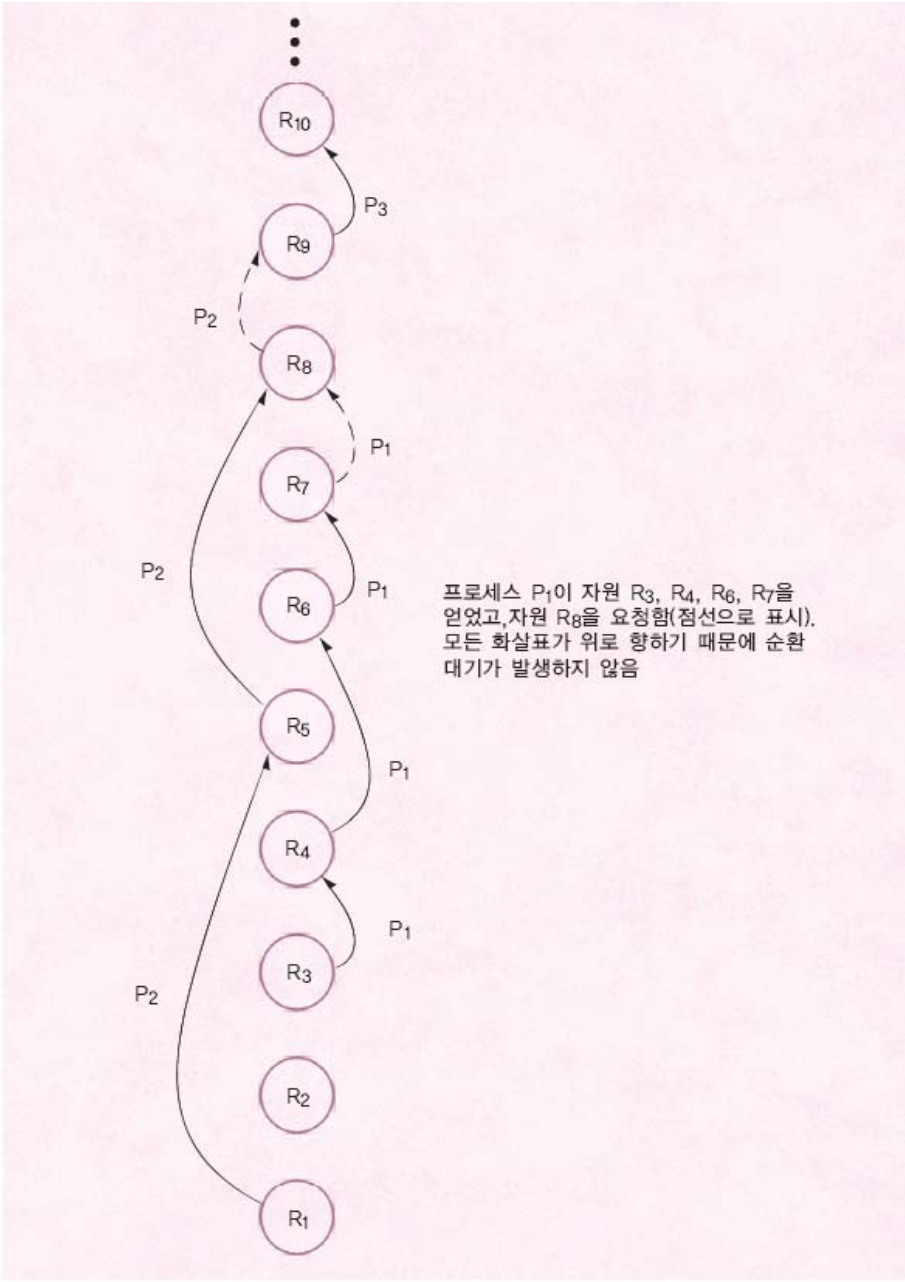
교착 상태 해결책

- 교착 상태 방지(예방)
- 교착 상태 회피
- 교착 상태 탐지
- 교착 상태 복구

교착 상태 방지

- 교착 상태 방지(예방)
  - 4가지의 교착상태 조건 중 하나라도 발생하지 않도록 함
  - 상호배제 문제는 고려해야 함
    - 전용자원에서는 교착상태가 발생하지 않으므로, 프로세스가 원하는 자원을 배타적으로 사용하려는 것은 제외시켜야 함
  - '점유와 대기' 조건 거부
    - 최대 자원 할당
      - 프로세스가 작업을 수행하기 전에 필요한 모든 자원을 요청하고 획득해야 함
      - 보류 상태에서는 프로세스가 자원을 점유할 수 없으므로 대기 조건이 성립하지 않음
    - 자원의 효율성이 낮음 (많은 자원이 사용되지 않으면서 오랜 시간 할당되기 때문)
  - '비선점' 조건 거부
    - 자원을 보유한 프로세스는 추가적인 자원 요청을 할 수 없다
    - 해당 프로세스는 일단 보유한 자원을 반납한 후 추가로 필요한 자원과 함께 한꺼번에 다시 요청해야 한다 (시스템에 의해 선점 당하게 됨)
    - 프로세스가 재 시작 되어야 하는 추가적인 부하 발생
  - '순환 대기'조건 거부
    - 자원에 대한 선형 순서 사용
    - 다른 방법에 비해 효율적인 자원 사용량
    - 자원의 요구에 대해 동적이거나 유연하지 못함

- ‘순환 대기’ 조건 거부
  - 자원에 대한 선형순서(linear ordering)
    - 시스템이 관리하는 프린터, 스캐너, 파일 등 자원마다 고유 번호 할당
  - 프로세스는 오름차순을 지키며 필요한 자원을 요청해야 함
    - 예상된 순서와 다르게 자원을 요구하는 작업은 자원 낭비를 초래함



[그림 7-3] 교착 상태 방지를 위한 하벤더의 선형 자원 순서

다익스트라의 은행원 알고리즘을 사용한 교착 상태 회피

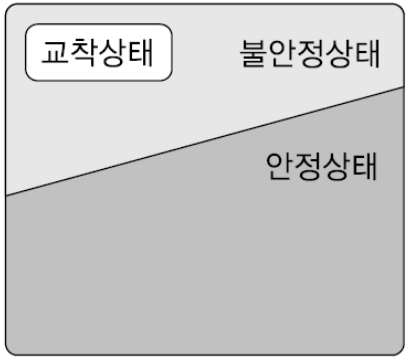
- 교착상태 회피
  - 교착상태의 예방보다 덜 엄격한 조건을 요구함으로써 자원을 좀 더 효율적으로 이용하는 것을 목적으로 함

- 은행원 알고리즘(Banker's Algorithm)
  - 프로세스가 요청한 자원을 할당했을 때, 교착상태가 발생할 수 있다면 요청한 자원을 할당하지 않음
  - 시스템은 관리하는 모든 자원을 자원 유형으로 그룹
  - 안정 상태
    - 운영체제는 모든 현재의 프로세스는 그들의 작업을 유한 시간 내에 완료되도록 보장
  - 불안정 상태
    - 시스템이 교착 상태에 있지 않지만, 현재의 프로세스의 작업을 유한 시간 내에 완료되는 것을 보장하지 못함

[참고] 안정상태와 불안정상태

- 안정상태와 불안정상태
  - 교착상태 회피 알고리즘은 시스템이 순환-대기 조건이 발생하지 않도록 자원 할당 상태를 검사함
  - 자원 할당 상태는 사용 가능한 자원의 수, 할당된 자원의 수, 프로세스들의 최대 요구 수에 의해 정의됨
  - 안정한 상태
    - 각 프로세스에 자원을 할당할 수 있고(최대치까지), 교착상태를 방지할 수 있음
    - 프로세스의 순서 $<P_1, P_2, \dots, P_n>$이 안정 순서란 의미는 모든 P_i가 요청하는 자원이 현재 사용 가능한 자원과 $j > i$인 모든 P_j가 점유하고 있는 자원들로 충족될 수 있음을 나타냄
    - 프로세스 P_i가 필요한 자원을 즉시 사용할 수 없다면, P_i는 모든 P_j가 끝날 때까지 기다렸다가 자원을 확보, 필요한 모든 자원 확보 시 지정된 작업을 끝내고 자원을 반납함
    - P_i가 종료하면 P_{i+1}은 필요한 자원을 확보할 수 있으므로 처리를 계속 진행할 수 있음
  - 불안정한 상태
    - 안정한 상태처럼 프로세스의 자원 할당 및 해제의 순서가 명확히 존재하지 않는 경우

※ 교착상태는 불안정상태이나, 모든 불안정상태가 교착상태인 것은 아님



안정상태, 불안정상태와 교착상태의 공간

[참고] 안정상태와 불안정상태

- 시스템 상태 변화
  - 동일한 자원 12개와 프로세스 P_0, P_1, P_2를 가진 시스템의 경우를 예를 들어 시스템 상태 변환을 설명함
  - 시스템의 안정 상태
    - 프로세스 P_0은 자원을 10개 요구, P_1은 4개, P_2는 9개 요구함
    - t_0 시간에 프로세스 P_0가 자원을 5개 점유, 프로세스 P_1은 2개, 프로세스 P_2는 2개를 점유할 경우 사용 가능한 자원은 3개임

[표1] 안정상태의 자원 예(1)

구분	현재 사용량	최대 사용량
P_0	5	10
P_1	2	4
P_2	2	9
사용 가능한 자원 수		3

[참고] 안정상태와 불안정상태

- 실행 과정
  - t_0 시간에 시스템은 안정상태이며, $<P_1, P_0, P_2>$ 순서는 안정 조건을 만족함
  - 프로세스 P_1은 사용 가능한 자원을 2개 할당 받아 실행한 후 반납 가능하므로 시스템의 여분 자원은 5개임
  - P_0은 사용 가능한 자원 5개를 할당 받아 실행한 후 반납 가능함
  - 프로세스 P_2가 필요한 자원을 할당받고 실행한 후 반납 가능함

[표2] 안정상태의 자원 예(2)

구분	초기	P_1	P_1 실행 후	P_0	P_0 실행 후	P_2	P_2 실행 후
현재 사용량	–	2	0	5	0	2	0
할당량	–	2	0	5	0	7	0
최대 사용량	–	4	0	10	0	9	0
사용 가능한 자원	3	1	5	0	10	3	12

- 불안정상태
  - 사용 가능한 자원 1개를 어느 프로세스에 할당해도 프로세스를 만족시킬 수 없음
  - 프로세스 P_0에 남은 장치를 할당하고 반납 전까지 다른 프로세스가 자원을 요구하지 않는 경우 교착상태를 피할 수 있음

[표3] 불안정상태의 자원 예(1)

구분	현재 사용량	최대 사용량
P_0	8	10
P_1	2	4
P_2	1	9
사용 가능한 자원 수		1

[참고] 안정상태와 불안정상태

- 안정상태에서 불안정상태로의 변환
  - [표 1]에서 프로세스 $\mathbf{P_2}$가 자원을 1개 더 요구할 경우, 이를 허용 시 불안정상태로 변함

[표4] 불안정상태의 자원 예(2)

구분	현재 사용량	최대 사용량
P_0	5	10
P_1	2	4
P_2	3	9
사용 가능한 자원 수		2

- 실행 과정
  - 프로세스 P_1은 사용 가능한 자원 2개를 할당 받아 실행한 후 장치를 반환
  - 프로세스 P_0는 자원 5개를 할당 받았지만 최대 10개가 필요하므로 5개를 더 요청하나, 최대 사용량만큼 자원을 할당할 수 없으므로 대기
  - 프로세스 P_2는 추가로 자원을 6개 요청하므로, 시스템은 교착상태가 됨
- 해결 방법
  - 다른 프로세스들이 처리를 종료하고 자원을 반납할 때까지 프로세스 P_2를 대기시키면 교착상태를 회피할 수 있음
- ※ 시스템이 항상 안정상태에 머무르도록 안정상태 개념에서 교착상태 회피 알고리즘을 정의 가능
- ※ 초기 시스템은 안정상태이므로, 프로세스가 현재 사용 가능한 자원을 요청 시 시스템은 자원 할당/대기 여부를 결정, 자원을 할당한 이후에도 시스템이 항상 안정상태일 경우에만 할당을 허용함

[참고] 교착상태 회피: 은행원 알고리즘

- 은행원 알고리즘
  - 다익스트라의 은행원 알고리즘(Banker's Algorithm)을 이용
    - 각 프로세스가 요청하는 자원 종류의 최대수를 알아야 함
    - 각 프로세스에 자원을 어떻게 할당(자원 할당 순서 조정)할 것인가의 정보가 필요하며, 이를 이용하여 교착상태 회피 알고리즘을 정의함
    - 은행에서 모든 고객이 만족하도록 현금을 할당하는 과정과 동일함
  - 구현 방법
    - 구현을 위해 여러가지 자료구조를 유지해야 하며, 이는 자원 할당 시스템의 상태를 나타냄
    - **n**은 시스템의 프로세스 수, **m**을 자원 형태의 수라 가정할 때, 다음과 같은 자료구조가 필요함

      - **Available**

        * 각 형태별로 사용 가능한 자원의 수를 표시하는 길이가 **m**인 벡터

      - **Max**

        * 각 프로세스의 최대 자원의 요구를 표시하는 **n x m** 행렬

      - **Allocation**

        * 현재 각 프로세스에 할당되어 있는 각 형태의 자원 수를 정의하는 **n x m** 행렬

      - **Need**

        * 각 프로세스에 남아 있는 자원 요구를 표시하는 **n x m** 행렬

[참고] 교착상태 회피: 은행원 알고리즘

- 구현을 위한 제약
  - 간단한 구현을 위해 다음과 같은 제약을 둠
    ① 시간이 진행하면서 벡터의 크기와 값이 변한다
    ② **X**와 **Y**의 길이가 n인 벡터이다
    ③ **X[i] ≤ Y[i]** 이고, **i** = 1, 2, …, **n**일 경우에만 **X ≤ Y**다
    ④ **X** = (0, 3, 2, 1)이고 **Y** = (1, 7, 3, 2)이면 **X ≤ Y**다
    ⑤ **X ≤ Y** 이고 **X ≠ Y** 이면 **X < Y**다

  - 행렬 **Allocation**과 **Need**에 있는 각 행을 벡터로 취급하며 이들을 각각 $\mathbf{Allocation}_i$와 $\mathbf{Need}_i$로 참조함
    - $\mathbf{Allocation}_i$는 프로세스 $\mathbf{P}_i$에 현재 할당된 자원
    - $\mathbf{Need}_i$는 프로세스 $\mathbf{P}_i$가 자신의 작업을 종료하는데 필요한 추가 자원

[참고] 교착상태 회피: 은행원 알고리즘

- 은행원 알고리즘
  - $\mathbf{Request_i}$를 프로세스 $\mathbf{P_i}$를 위한 요청 벡터라 가정함
  - 만약 '$\mathbf{Request_i[j] = k}$'라면, 프로세스 $\mathbf{P_i}$는 자원 형태 $\mathbf{R_j}$를 $\mathbf{k}$개 요구함
  - 프로세스 $\mathbf{P_i}$가 자원을 요청 시 다음 동작이 일어남

    * 1단계 : $\mathbf{Request_i \le Need_i}$면 2단계로 가고, 그렇지 않으면 프로세스가 최대 요구치를 초과하기 때문에 오류 상태가 됨

    * 2단계 : $\mathbf{Request_i \le Available}$면 3단계로 가고, 그렇지 않으면 자원이 부족하기 대문에 $\mathbf{P_i}$는 대기

    * 3단계 : 시스템은 상태를 다음과 같이 수정하여 요청된 자원을 프로세스 $\mathbf{P_i}$에 할당

      - $\mathbf{Available := Available - Request_i;}$
      - $\mathbf{Allocation_i := Allocation_i + Request_i;}$
      - $\mathbf{Need_i := Need_i - Request_i;}$

  - 자원 할당 상태가 안정이라면 처리가 이루어지고 프로세스 $\mathbf{P_i}$는 자원을 할당 받음
  - 불안정 상태이면 $\mathbf{P_i}$는 $\mathbf{Request_i}$를 대기하고 이전 자원 할당 상태로 복귀함

다익스트라의 은행원 알고리즘을 사용한 교착 상태 회피

- 은행원 알고리즘
  - 안정 상태 예

[표 7-1] 안전 상태

Max Allocation Need

프로세스	max(P_i) (최대 필요 수)	loan(P_i) (현재 대여 수)	claim(P_i) (현재 요청 수)
P_1	4	1	3
P_2	6	4	2
P_3	8	5	3
총 자원 수 t = 12		가용 자원 수 a = 2	

Available

다익스트라의 은행원 알고리즘을 사용한 교착 상태 회피

- 은행원 알고리즘

[표 7-2] 불안전 상태

프로세스	max(P_i) (최대 필요 수)	loan(P_i) (현재 대여 수)	claim(P_i) (현재 요청 수)
P_1	10	8	2
P_2	5	2	3
P_3	3	1	2
총 자원 수 t = 12		가용 자원 수 a = 1	

다익스트라의 은행원 알고리즘을 사용한 교착 상태 회피

- 은행원 알고리즘
  - 안정 상태에서 불안정 상태로의 전이 ([표 7-1] → [표 7-3])
    - [표 7-1]에서, 현재 a값은 2

[표 7-3] 안전 상태에서 불안전 상태로 전이

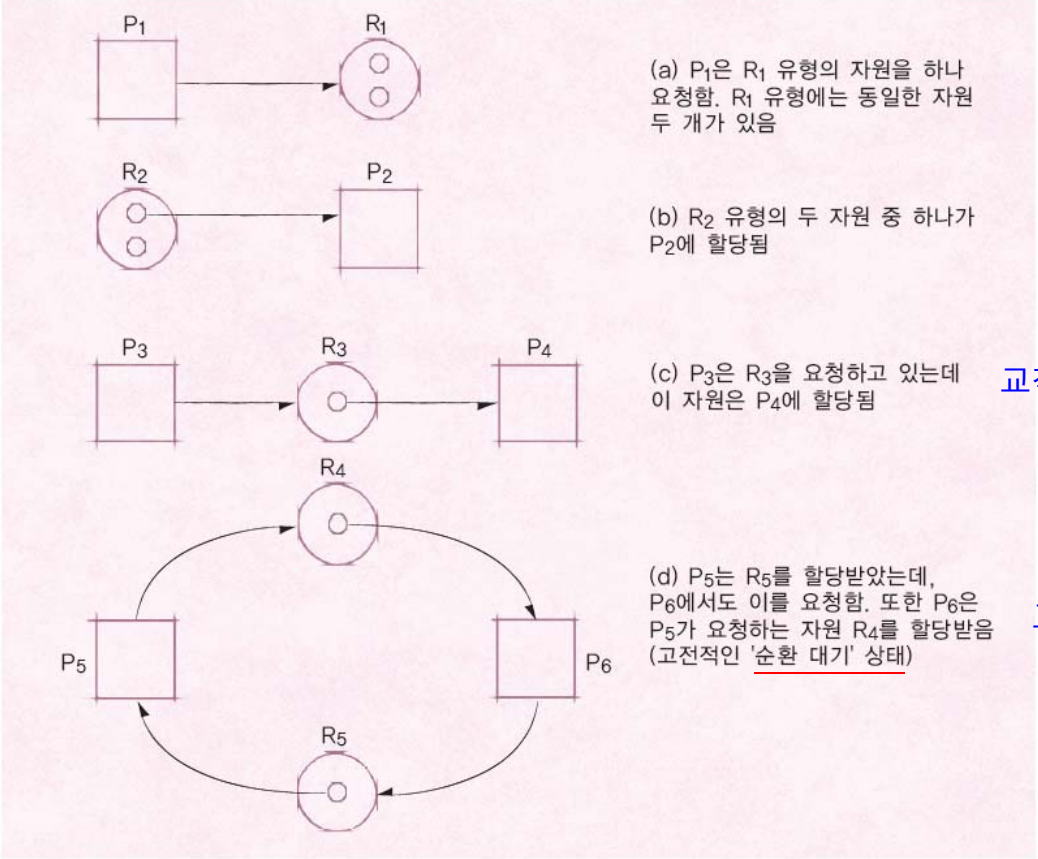
프로세스	max(P_i) (최대 필요 수)	loan(P_i) (현재 대여 수)	claim(P_i) (현재 요청 수)
P_1	4	1	3
P_2	6	4	2
P_3	8	6	2
총 자원 수 t = 12		가용 자원 수 a = 1	

교착 상태 탐지

- 교착 상태 존재 여부
- 교착 상태와 연관된 프로세스와 자원을 알아내는 과정
- 순환 대기 존재 여부에 초점
- 실행 시 심각한 부담을 줄 수 있음
  - Trade-off문제

교착 상태 탐지

- 자원 할당 그래프
  - 정사각형 : 프로세스
  - 큰 원 : 같은 유형의 자원이 들어 있는 집합
  - 작은 원 : 특정 유형 안에 있는 각 자원



교착상태 발생 가능한 상황

교착상태 발생

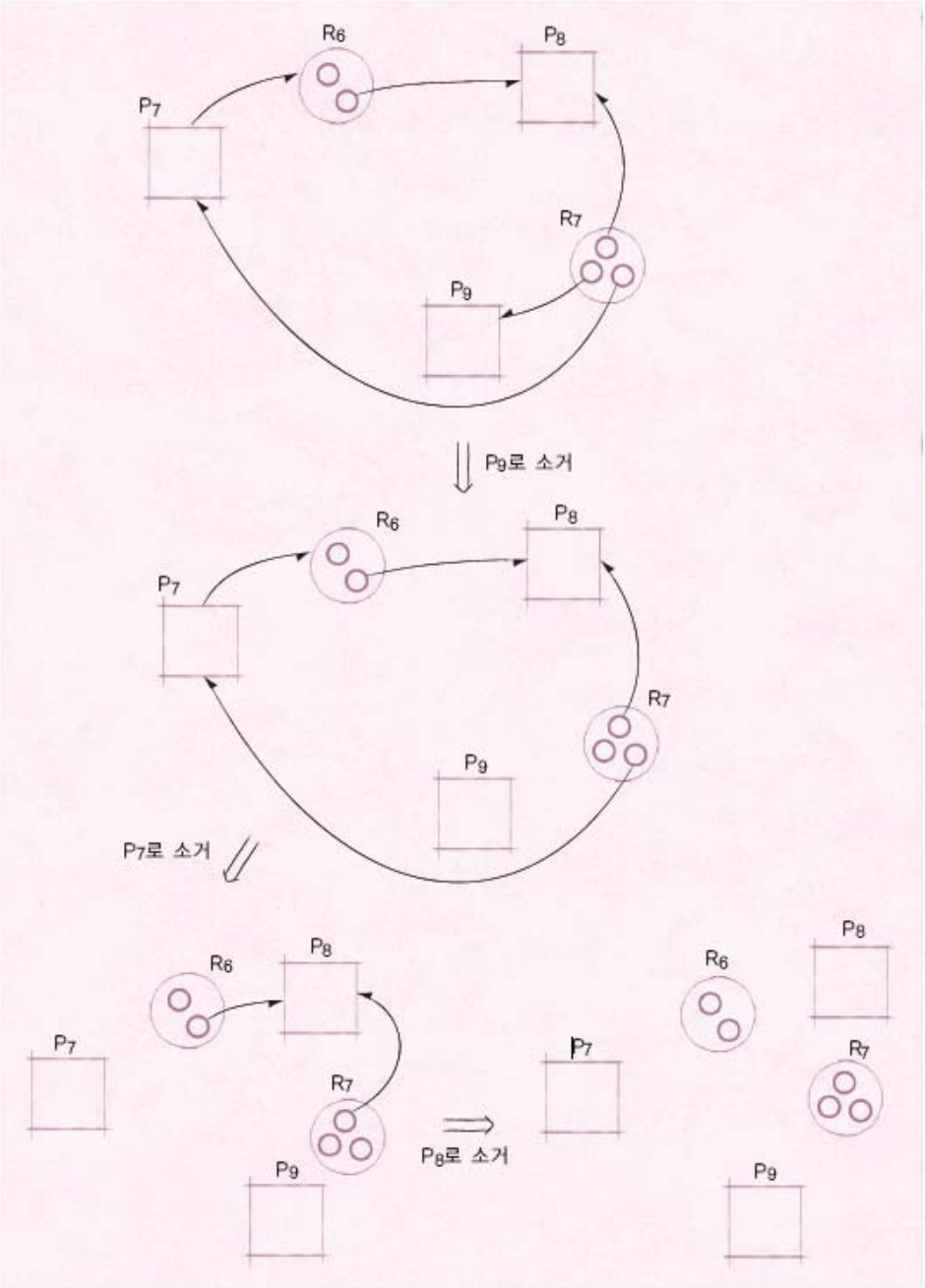
[그림 7-4] 자원 할당 및 요청 그래프

교착 상태 탐지

- 자원 할당 그래프 소거
  - 실행을 완료할 수 있는 그래프가 있는지, 교착 상태의 프로세스가 있는지 확인
  - 한 프로세스의 자원 요청을 허용할 수 있다면 해당 프로세스를 통해 그래프 소거
  - 그래프가 모든 프로세스에 대해 소거될 수 있으면 교착 상태는 없음

교착 상태 탐지

- 자원 할당 그래프 소거



[그림 7-5] 그래프 소거를 통해 교착 상태가 존재하지 않음을 증명하는 방법

교착 상태 복구

- 교착 상태 복구
  - 특정 프로세스를 강제로 제거하고 자원을 반납하라고 요구하는 것
- 일시 정시/재시작 메커니즘
  - 시스템이 프로세스를 일시적으로 정지하고 안전 상태가 되었을 때 해당 프로세스를 작업 손실 없이 놓아주는 것
- 체크 포인트/롤백
  - 시스템 다운이나 교착 상태에 대처하기 위해 종료되는 각 프로세스로부터 가능하면 많은 양의 데이터를 보존하는 기능
    - 작업 손실의 양을 마지막 체크포인트(check point, 시스템을 정상적으로 저장한 상태)까지로 제한