Slide 0

2부 프로세스 관리
(8장. 교착상태)

시스템 모델

Slide 1

- 시스템 구성요소
  - 자원
    - 유한한 수의 자원
    - 예) 주기억장치 공간, CPU 주기, 입출력 장치, 파일등
  - 프로세스
    - 여러 프로세스가 자원 획득을 위해 경쟁함
    - 자원의 수보다 자원을 요청한 프로세스가 많은 경우 대기하여야함
- 자원의 사용
  - 요청
    - 자원의 사용을 요청함
    - 요청이 즉시 허용되지 않으면 대기하여야함
    - 다른 프로세스가 해당 자원을 사용할 경우 대기하여야함
  - 사용
    - 자원 사용이 허락 되면 자원을 사용, 예) 프린터로 자료 출력
  - 해제
    - 자원 사용이 모두 끝나면 다른 프로세스를 위해 사용을 해제함

시스템 모델 (계속)

- 자원 요청과 해제
  - 자원의 요청과 해제는 시스템 호출을 통하여 수행
  - 자원의 대기는 세마포를 이용하여 구현
- 교착상태

Slide 2

  - 발생
    - 어떤 집합내에 있는 모든 프로세스가
    - 그 집합내에 있는 다른 프로세스가 사용중인 자원을 기다리고 있는 경우
  - 예)
    - 세 개의 테이프드라이브를 가진 시스템에서
      세개의 프로세스가 각각의 테이프 사용중 다른 테이프의 사용을 요청
    - 한 프로세스는 테이프를 다른 프로세스는 프린터 사용 중
      서로 상대방이 사용하는 장치를 요청한 경우

교착상태의 특징

- 필요 조건 (아래 4가지 조건을 모두 만족하면 교착상태 발생)
  - 상호배제 (mutual exclusion)
    - 최소한 하나의 자원이 비공유 방식으로 점유
    - 비공유 $\rightarrow$한 번에 하나의 프로세스만 자원을 사용함을 의미
  - 점유와 대기 (hold & wait)
    - 프로세스는 최소한 하나의 자원을 점유
    - 이 프로세스는 다른 프로세스가 점유하고 있는 자원을 얻기위해 대기

Slide 3

  - 비선점 (no preemption)
    - 자원은 강제로 해제될 수 없음
    - 프로세스가 작업을 종료한 후에 자원이 해제됨
  - 순환대기 (circular wait)
    - 프로세스 집합 $\{P_0, P_1, \ldots, P_n\}$ 에서
    - P_0 는 P_1이 점유한 자원을 대기하고
    - P_1 는 P_2가 점유한 자원을 대기하고
    - P_{n-1} 는 P_n이 점유한 자원을 대기하고
    - P_n 는 P_0가 점유한 자원을 대기하는 것

교착상태의 특징 (계속)

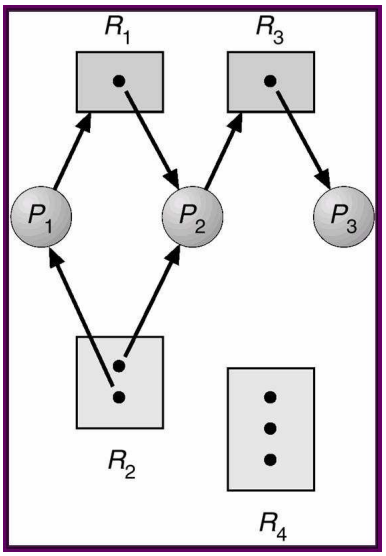
Slide 4

- 자원할당 그래프
  - 구성요소
    - 노드
      - 프로세스 노드: $P = \{P_1, P_2, \ldots, P_n\}$, 원으로 표시
      - 자원 노드: $R = \{R_1, R_2, \ldots, R_n\}$, 직사각형으로 표시 (직사각형 내부의 점은 여러개의 동일 자원을 의미)
    - 선
      - $P_i \rightarrow R_j$: 요청선으로 P_i가 자원 R_j를 요청함
      - $R_j \rightarrow P_i$: 할당선으로 R_j가 프로세스 P_i에 할당됨
  - 주기 (cycle)
    - 자원할당 그래프에서 요청선과 할당선이 하나의 cycle 을 이루는 것
    - 자원이 종류별로 하나씩 있는 상황에서 주기가 발생한 경우 →교착상태임

교착상태의 특징 (계속)

Slide 5

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

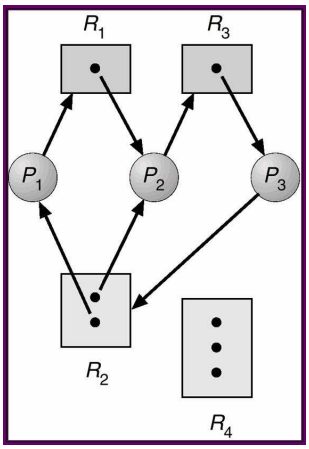


교착상태의 특징 (계속)

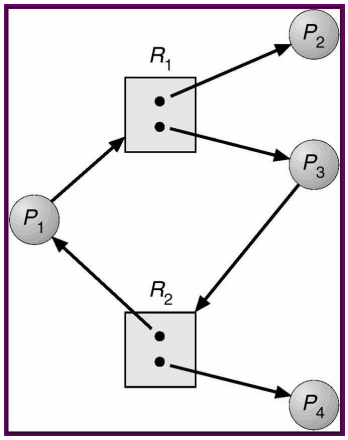
Slide 6

- 교착상태
  - 자원할당 그래프에 주기 (cycle)이 없으면 교착상태 아님
  - 자원할당 그래프에 주기 (cycle)이 있으면 교착상태 가능성 있음

* 주기가 있는 교착상태 예)



* 주기가 있으나 교착상태 없는 예)



교착상태 처리 방법

Slide 7

- 세 가지 방법
  - 교착상태가 일어나지 않도록 프로토콜을 사용
    - 교착상태 예방(prevention) 방법
    - 교착상태 회피(avoidance) 방법
  - 교착상태를 허용하나 교착상태를 탐지하여 제거
    - 교착상태 탐지(detection) 방법
  - 교착상태를 무시하는 방법
    - Unix를 포함한 대부분의 운영체제가 사용하는 방법
    - 교착상태는 보통 1년에 한 번정도 발생
    - 발생시는 재시작 해야함
    - 적은 비용이 장점

교착상태 예방

Slide 8

- 설명
  - 교착상태를 발생시키지 않기 위하여
  - 교착상태 발생 4가지 조건 중 하나를 만족시키지 않는 방법
- 상호배제 조건
  - 상호배제가 되지 않아도 되도록 함
    - 상호배제란 다른 의미로 자원의 비공유를 의미
    - 즉 자원의 공유를 허용해서 해결
    - 어떤 자원은 공유 가능 →읽기 전용의 파일
    - 어떤 자원은 근본적으로 공유 불가능 →파일 쓰기등
  - 결론: 상호배제를 통하여 교착상태를 예방하기 어려움

교착상태 예방 (계속)

Slide 9

- 점유와 대기 조건
  - 한 자원을 점유하면서 다른 자원을 대기할 수 없도록 하는 두 가지 방법
    - 실행되기 전에 해당 프로세스가 필요로 하는 모든 자원을 점유하게 함
    - 실행 중 대기가 발생하지 않음
    - 실제 필요 전에 할당 받음 (문제: 사용율이 낮아짐)
    - 필요로하는 자원 중 하나라도 다른 프로세스가 점유하면 대기함 (문제: 기아 발생 가능)
  - 한 자원을 점유한 상태에서 다른 자원을 요청할 수 없게 함
    - 다른 자원을 요청하려면 반드시 점유한 자원을 먼저 해제해야함 (즉, 점유가 없을 때만 요청이 가능)

교착상태 예방 (계속)

- 비선점
  - 비선점 조건을 제거하여 해결 →즉 선점 (해제)를 허용
  - 방법 1
    - 자원을 점유하고 있는 프로세스가 대기해야하는 다른 자원을 요구한 경우
    - 그 프로세스가 점유하고 있는 모든 자원을 선점(해제) 시킴
    - 그 프로세스는 요구 자원뿐만 아니라 해제한 자원을 획득하면 재시작 됨

Slide 10

  - 방법 2
    - $P_i \rightarrow R_i$ 이고 $R_i \rightarrow P_j$ 이고 $P_j \rightarrow R_j$ 이면
    - $R_i \rightarrow P_j$ 를 선점 (해제) 하고
    - $R_i \rightarrow P_i$ 함
  - 제한
    - 이 방법들은 CPU, 레지스터, 기억장치공간등과 같이 상태가 쉽게 기억되고 재사용이 가능한 자원에만 적용됨
    - 프린터, 테이프 구동기 같은 자원에는 적용할 수 없음
      (만약 프린터에 적용하면 출력된 인쇄물이 이 내용 저 내용이 섞이게 됨)

교착상태 예방 (계속)

- 순환 대기
  - 순환대기의 발생을 막음으로서 해결
    - 자원집합: $R = \{R_1,\ R_2, \ldots,\ R_m\}$
    - 함수정의
      - $F:\ R \rightarrow N$
      - 예)

Slide 11

        F(tape drive)=1,
        F(disk drive)=5,
        F(printer)=12.
  - 방법 1
    - 프로세스 P_i 가 R_i 자원을 점유하고 있으면
    - 이 프로세스는 $F(R_j) > F(R_i)$ 인 자원만 요청 가능
  - 방법 2
    - 프로세스 P_i 가 R_i 자원을 요청하기 전에
    - 이 프로세스는 $F(R_j) < F(R_i)$ 인 모든 자원을 해제

교착상태 회피

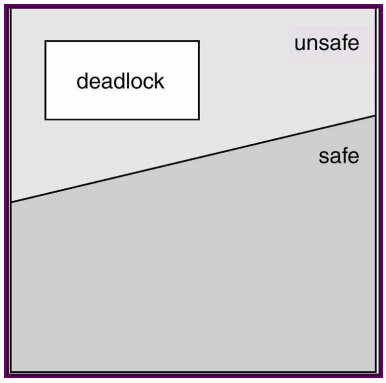
Slide 12

- 설명
  - 자원이 어떻게 요청되는지에 대한 추가적인 정보를 이용해 교착상태를 피하는 방법
  - 문제점: 부수적인 정보의 유지와 교착상태를 피하기 위한 알고리즘은 운영체제에 부담
- 방법
  - 자원들에 대한 요청이 들어올 때마다
    - 현재 사용 가능한 자원들
    - 각 프로세스에 할당된 자원들
    - 할당되었으나 반환될 자원들을 검사하여
    - 지금 들어온 요청을 허가하고도 교착상태에 빠지지 않을 것인가를 확인한 후에 결정

교착상태 회피 (계속)

Slide 13

- 안전상태 (safe state)
  - 교착상태가 발생할 수 없는 상태
  - 안전순서 (safe sequence) 가 존재하면 안전 함
  - 그렇지 않으면 불안전 함 (불안전 하다고 교착상태는 아님)



  - 불안전 상태에서 교착상태 예방은 불가능
  - 그러므로 자원 요청시 안전상태를 유지하는지 점검 후 할당

교착상태 회피 (계속)

- 안전순서 (safe sequence)
  - 다음 조건을 만족하는 프로세스의 순서 $< P_1,\ P_2, \ldots,\ P_n >$ 이 존재
  - 조건
    - 각 P_i 요청에 대해
    - 현재 사용 가능한 자원과 $j < i$ 인 P_j가 점유하고 있는 자원에 의해 만족

Slide 14

- 예)
  - 테이프 장치: 12개
  - 프로세스: 3개 ($P_0,\ P_1,\ P_2$)
  - 최대 수요 및 현재 수요

	최대 수요	현재 수요
P_0	10	5
P_1	4	2
P_2	9	2

교착상태 회피 (계속)

  - 시간 t_0에서 현재 수요 할당
    - 안전상태
    - $< P_1, P_0, P_2 >$의 안전순서 존재하므로
  - 시간 t_1에서 P_2가 하나 더 요청
    - 이것을 허락하면 불안전 상태가 됨

Slide 15

	최대 수요	현재 수요	최대 요청
P_0	10	5	5
P_1	4	2	2
P_2	9	3	6

    - P_1 은 2개의 여분 자원을 이용해 할당 받고 해제 가능
    - 단, 이후 4개의 여분으로 P_0 5개 와 P_2 6개 모두 할당 불가능 함 →교착상태
  - 해결
    - t_1에서 P_2의 요청에 대해 안전상태 유지가 불가능함으로 허용하지 않고 대기
    - P_2의 요청을 허용해도 안전상태가 유지될때 허용

교착상태 회피 (계속)

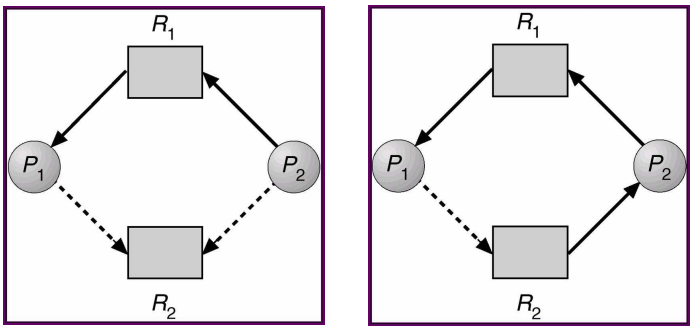
Slide 16

- 자원할당 그래프 알고리즘
  - 각 자원이 한개씩 있을 때 사용가능한 알고리즘
  - 기존 자원할당 그래프에 요청가능선(claim edge) 점선으로 표시
  - 실제 요청이 일어나면 요청가능선에 대하여 할당시 주기가 발생하는지 검사 후 발생하지 않으면 허용 (이때 요청가능선도 포함하여 검사)

교착상태 회피 (계속)

Slide 17

- 예)



  * P_2가 R_2를 요청할 경우
    · 할당하면 주기 발생
    · 그러므로 할당하지 않음
  * P_1이 R_2를 요청할 경우
    · 할당해도 주기 발생하지 않음
    · 그러므로 할당

교착상태 회피 (계속)

Slide 18

- 은행가 알고리즘 (Banker's algorithm)
  - 설명
    - 자원할당 그래프 알고리즘에 반해 자원별로 다수개 있을 때 적용 가능
    - 자원할당 그래프 알고리즘보다는 덜 효과적
    - 각 프로세스는 각 자원의 필요 최대수를 선언해야하며 이것이 각 자원의 최대수를 초과하지 않아야함
  - 자원 할당
    - 자원요구시 자원할당 후에도 안전상태를 유지하는지 점검후
    - 안전상태 유지하면 할당, 그렇지 않으면 할당하지 않음
    - 자원을 할당받지 못한 프로세스는 대기함

교착상태 회피 (계속)

Slide 19

  - 알고리즘 사용 데이터 구조
    - *Available*
      - 이용 가능한 자원별 수
      - *Available[j]* = *k*: R_j의 자원이 k개 사용가능함을 의미
    - *Max*
      - 프로세스가 요구하는 자원의 최대수를 $n \times m$ 행렬로 나타냄
      - *Max[i,j]* = *k*: P_i 프로세스는 자원 R_j를 최대 k개까지 요구
    - *Allocation*
      - 각 프로세스에게 할당된 자원의 수를 $n \times m$ 행렬로 나타냄
      - *Allocation[i,j]* = *k*: P_i 프로세스에 자원 R_j가 k 개 할당됨
    - *Need*
      - 각 프로세스가 추가로 필요로하는 자원의 수를 $n \times m$ 행렬로 나타냄
      - *Need[i,j]* = *Max[i,j]* - *Allocation[i,j]*
  - 간략 표현
    - 아래 조건을 만족할 때 $X \leq Y$ 라고 표기함
      - $X[i] \leq Y[i]$, for all $i = 1, 2, \ldots, n$
      - 예) $X = (1, 7, 3, 2)$ 이고 $Y = (0, 3, 2, 1)$ 이면 $Y \leq X$ 임

교착상태 회피 (계속)

Slide 20

- 안전상태 검사 알고리즘
  1. 초기화
     * $Work := Available$
     * $Finished[i] := false, i = 1, 2, \ldots, n$
  2. 다음 조건을 만족하는 것을 찾음
     * $Finish[i] = false$ 와 $Need_i \leq Work$
     * 만족하는 것이 없으면 4로
  3. 다음을 수행
     * $Work := Work + Allocation_i$ 과 $Finish[i] := true$
     * 2로
  4. 다음 조건을 만족하면 안전 상태임
     * $Finish[i] = true$ for all i

교착상태 회피 (계속)

Slide 21

- 자원요청 알고리즘
  1. 요청
     * $Request_i \leq Need_i$ 이면 2로
     * 아니면 오류
  2. 검사
     * $Request_i \leq Available$ 이면 3으로
     * 아니면 프로세스 대기
  3. 임시로 계산후 할당
     * $Available := Available - Request_i$
     * $Allocation_i := Allocation_i + Request_i$
     * $Need_i := Need_i - Request_i$
     * 계산후 안전상태이면 할당
     * 아니면 할당하지 않음

교착상태 회피 (계속)

Slide 22

- 예)
  - 프로세스: $P_0, \ldots, P_4$
  - 자원: A(10개), B(5개), C(7개)
  - 시간 t_0에서 상황

	Allocation	Max	Available	Need
	A B C	A B C	A B C	A B C
P0	0 1 0	7 5 3	3 3 2	7 4 3
P1	2 0 0	3 2 2		1 2 2
P2	3 0 2	9 0 2		6 0 0
P3	2 1 1	2 2 2		0 1 1
P4	0 0 2	4 3 3		4 3 1

➥ 이 상황에서 안전함: 안전순서 $\rightarrow < P_1, P_3, P_4, P_2, P_0 >$

교착상태 회피 (계속)

Slide 23

- P_1 요청: $Request_1 = (1, 0, 2)$
  - 검사: $Request_1 \leq Available \rightarrow$(1,0,2) $\leq$ (3,3,2) 통과
  - 임시로 계산

	Allocation	Need	Available
	A B C	A B C	A B C
P0	0 1 0	7 4 3	2 3 0
P1	3 0 2	0 2 0	
P2	3 0 2	6 0 0	
P3	2 1 1	0 1 1	
P4	0 0 2	4 3 1	

  - 안전검사 알고리즘 수행 $\rightarrow$안전
  - 안전순서: $< P_1, P_3, P_4, P_0, P_2 >$

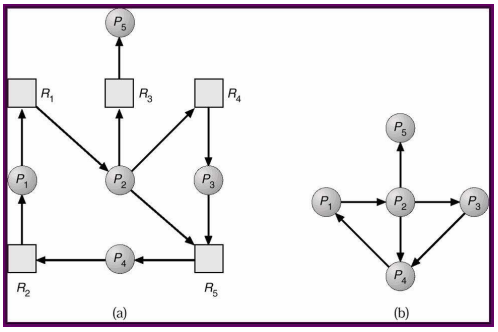
➥ 그러므로 P_1에게 해당 자원을 할당

  - P_4 (3,3,0)은 할당되나?
  - P_0 (0,2,0)은 할당되나?

교착상태 탐지

Slide 24

- 설명
  - 교착상태를 허용하고 주기적으로 교착상태를 발견하여 복구하는 방법
  - 교착상태 탐지를 위한 정보를 유지해야하고 복구과정에도 비용이 발생함
- 자원이 종류별로 한개씩 있는 경우
  - 자원할당 그래프를 대기그래프 (wait-for graph)로 변환
  - 변환: 요청선 $P_i \rightarrow R_q$ 와 할당선 $R_q \rightarrow P_j$ 를 대기선 $P_i \rightarrow P_j$로 변환



  - 교착상태 탐지 $\rightarrow$대기 그래프에 주기 탐지

교착상태 탐지 (계속)

Slide 25

- 자원이 종류별로 다수개 있는 경우
  - 은행가 알고리즘과 유사한 알고리즘 사용
  - 알고리즘 사용 데이터 구조
    - *Available*: 이용 가능한 자원별 수를 길이 m의 벡터로 나타냄
    - *Allocation*:각 프로세스에게 할당된 자원의 수를 $n \times m$ 행렬로 나타냄
    - *Request*: 각 프로세스가 요청하는 자원의 수를 $n \times m$ 행렬로 나타냄
  - 관계 $X \leq Y$ 의 정의는 은행가 알고리즘과 동일

교착상태 탐지 (계속)

Slide 26

- 탐지 알고리즘
  1. 초기화
     * $Work := Available$
     * $Finished[i] := false, i = 1, 2, \ldots, n$
  2. 다음 조건을 만족하는 것을 찾음
     * $Finish[i] = false$ 와 $Request_i \leq Work$
     * 만족하는 것이 없으면 4로
  3. 다음을 수행
     * $Work := Work + Allocation_i$ 과 $Finish[i] := true$
     * 2로
  4. 다음 조건을 만족하면 P_i는 교착상태임
     * $Finish[i] = false$, for some i, $1 \leq i \leq n$

교착상태 탐지 (계속)

Slide 27

- 예)
  * 프로세스: $P_0, \ldots, P_4$
  * 자원: A(7개), B(2개), C(6개)
  * 시간 t_0에서 상황

	Allocation	Request	Available
	A B C	A B C	A B C
P0	0 1 0	0 0 0	0 0 0
P1	2 0 0	2 0 2	
P2	3 0 3	0 0 0	
P3	2 1 1	1 0 0	
P4	0 0 2	0 0 2	

➥ 이 상황은 교착상태 아님
  · $< P_0, P_2, P_3, P_1, P_4 >$ 의 순서로 모든 i에서 *Finish[i]*=true 가 됨
  · P_0 수행되면 $\rightarrow Work = (0, 1, 0)$
  · P_2 수행되면 $\rightarrow Work = (3, 1, 3)$

교착상태 탐지 (계속)

Slide 28

* P_2 요청: $Request_2 = (0, 0, 1)$
  · *Request* 행렬

	Request		
	A	B	C
P0	0	0	0
P1	2	0	2
P2	0	0	1
P3	1	0	0
P4	0	0	2

➥ 교착상태 발생: P_0 만 수행가능

교착상태 탐지 (계속)

Slide 29

- 탐지 알고리즘의 사용
  - 탐지 알고리즘의 사용 빈도를 결정하는 요소
    * 교착상태가 발생하는 빈도 →많을 수록 자주 탐지해야함
    * 교착상태가 발생했을때 영향받는 프로세스의 수 →많을 수록 자주
  - 탐지 알고리즘의 적용 시점
    * 바로 할당될 수 없는 자원을 요청할 때마다
    * 매 요청마다 →고 비용으로 현실성이 없음
    * 한 시간 마다 혹은 CPU 이용률이 40% 이하로 될 때마다
      (교착상태는 CPU 이용률을 저하 시킴)

교착상태로부터 회복

Slide 30

- 교착상태를 탐지 후 처리 방법
  - 운영자에게 통보하여 수동으로 처리
  - 자동처리 방법
    * 순환대기 중인 한 개 이상의 프로세스를 종료
    * 교착상태에 있는 프로세스에서 자원을 강제로 해제
- 프로세스 종료(termination)
  - 두 가지 방법
    * 교착상태 프로세스를 모두 종료
      · 확실하게 순환대기를 해결하지만 고 비용
      (프로세스 수행 부분을 나중에 다시 수행해야 함으로)
    * 순환대기 프로세스 중 하나를 종료
      · 비용은 적게드나 교착상태 해결이 안될 가능성이 있음
      · 교착상태 해결 안되면 프로세스를 추가로 종료

교착상태로부터 회복 (계속)

Slide 31

  - 프로세스 종료시 문제점
    * 파일 갱신중이었으면 파일이 부정확한 상태가 됨
    * 작업 중간에 종료한 경우 하드웨어를 재 초기화 하여야 함 (예로 프린터)
  - 순환대기 프로세스중 하나 종료시 고려사항
    * 어떤 프로세스를 종료해야 교착상태를 벗어나는지를 결정해야함
    * 그러한 프로세스가 여러 개이면 그 중 가장 경제적인 프로세스를 선택해야함
    * 가장 경제적인 프로세스를 선택하기 위해서는 여러가지 측면을 고려
      · 프로세스의 우선순위
      · 프로세스의 실행시간, 남은시간
      · 프로세스 점유 자원의 형태와 수
      · 프로세스가 종료되기 위해 필요한 자원의 수
      · 교착상태를 해결하는데 필요한 프로세스 수
      · 프로세스의 종류 (대화식(interactive) 인지 혹은 일괄식(batch)인지)

교착상태로부터 회복 (계속)

Slide 32

- 자원의 강제해제
  - 교착상태 해결을 위하여 프로세스로부터 자원을 강제로 해제하여 다른 프로세스에게 제공
  - 고려요소
    - 희생자 선택
      - 어떤 자원을 어떤 프로세스로부터 해제할 것인지 결정함
      - 프로세스 종료와 유사하게 비용을 계산하여 저비용으로 결정
    - 복귀 (rollback)
      - 자원이 해제당한 프로세스는 후에 정상적인 상황으로 복귀되어야함
      - 가장 단순한 방법은 프로세스 종료후 재시작하는 방법
      - 효과적인 방법은 필요한 만큼만 복귀 (단, 많은 정보를 유지해야함)
    - 기아상태
      - 비용 근거 자원해제는 매번 동일한 프로세스가 선택될 가능성이 높음
      - 이 프로세스가 기아상태에 빠지지 않고 계속 수행될 수 있도록
      - 희생자로 선택되는 횟수에 제한이 있어야함