8장. 교착상태 (2)

운영체제

목차

학습목표

- Mutex락을 사용할 때 어떻게 교착상태가 발생할 수 있는지 보여준다.
- 교착상태를 특징짓는 4가지 필수 조건을 정의한다.
- 자원할당 그래프에서 교착상태 상황을 식별한다.
- 교착상태 예방을 위한 4가지 방법을 평가한다.
- 교착상태 회피를 위한 은행가 알고리즘을 적용한다.
- 교착상태 감지 알고리즘을 적용한다.
- 교착상태에서 복구시키기 위한 접근법을 평가한다.

8.7 DEADLOCK DETECTION

8. Deadlocks

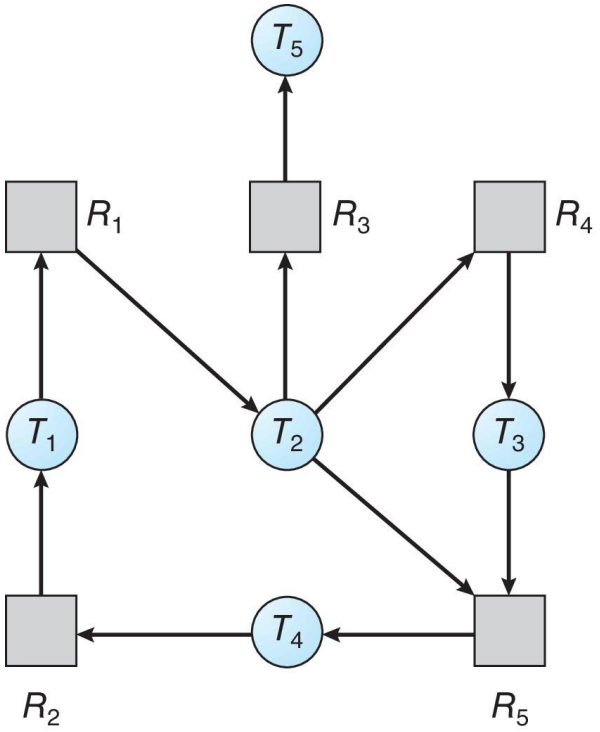
Deadlock Detection

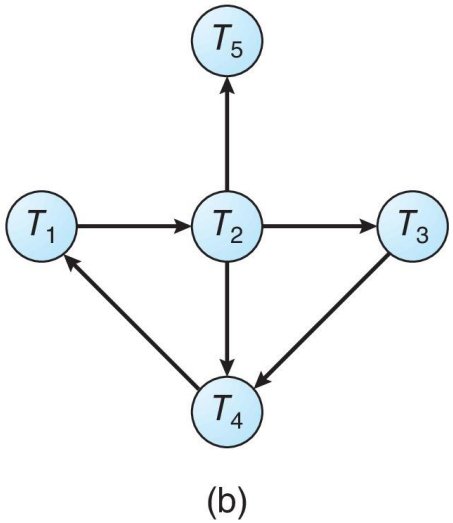
- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

8.7.1 Single Instance of Each Resource Type

- Maintain **wait-for** graph
  - Nodes are processes
  - $P_i \rightarrow P_j$ if P_i is waiting for P_j

Resource-Allocation Graph and Wait-for Graph





Resource-Allocation Graph

Corresponding wait-for graph

8.7.1 Single Instance of Each Resource Type

- Maintain **wait-for** graph
  - Nodes are processes
  - $P_i \rightarrow P_j$ if P_i is waiting for P_j

- Periodically invoke an algorithm that searches for a cycle in the graph.
  - If there is a cycle, there exists a deadlock

- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph

8.7.2 Several Instances of a Resource Type

- **Available**:
  - A vector of length ***m*** indicates the number of available resources of each type
- **Allocation**:
  - An ***n*** **x** ***m*** matrix defines the number of resources of each type currently allocated to each process
- **Request**:
  - An ***n*** **x** ***m*** matrix indicates the current request of each process.
  - If ***Request*[*i*][*j*] = *k***, then process P_i is requesting ***k*** more instances of resource type R_j.

Detection Algorithm

1. Let ***Work*** and ***Finish*** be vectors of length ***m*** and ***n***, respectively Initialize:
   a) ***Work*** = ***Available***
   b) For ***i*** = **1,2, …, *n***, if $\boldsymbol{Allocation_i \neq 0}$, then
      Finish**[i]** = ***false***; otherwise, ***Finish*****[i]** = ***true***

2. Find an index ***i*** such that both:
   a) ***Finish*****[*i*]** == ***false***
   b) $\boldsymbol{Request_i \leq Work}$

   If no such ***i*** exists, go to step 4

Detection Algorithm (Cont.)

3. $Work = Work + Allocation_i$
   $Finish[i] = true$
   go to step 2

4. If $Finish[i] == false$, for some i, $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if $Finish[i] == false$, then P_i is deadlocked

Algorithm requires an order of O(m x n^2) operations to detect whether the system is in deadlocked state

Example of Detection Algorithm

- Five processes P_0 through P_4; three resource types A (7 instances), *B* (2 instances), and *C* (6 instances)
- Snapshot at time T_0 :

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- Sequence <P_0,P_2,P_3,P_1,P_4> will result in ***Finish[i] = true*** for all ***i***

Example (Cont.)

- P_2 requests an additional instance of type C

	Allocation	Request	Available
	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 **1**	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- State of system?
  - Can reclaim resources held by process P_0, but insufficient resources to fulfill other processes; requests
  - Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4

8.7.3 Detection-Algorithm Usage

- When, and how often, to invoke depends on:
  - How often a deadlock is likely to occur?
  - How many processes will need to be rolled back?
    - One for each disjoint cycle
- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes “caused” the deadlock.

8.8 RECOVERY FROM DEADLOCK

8. Deadlocks

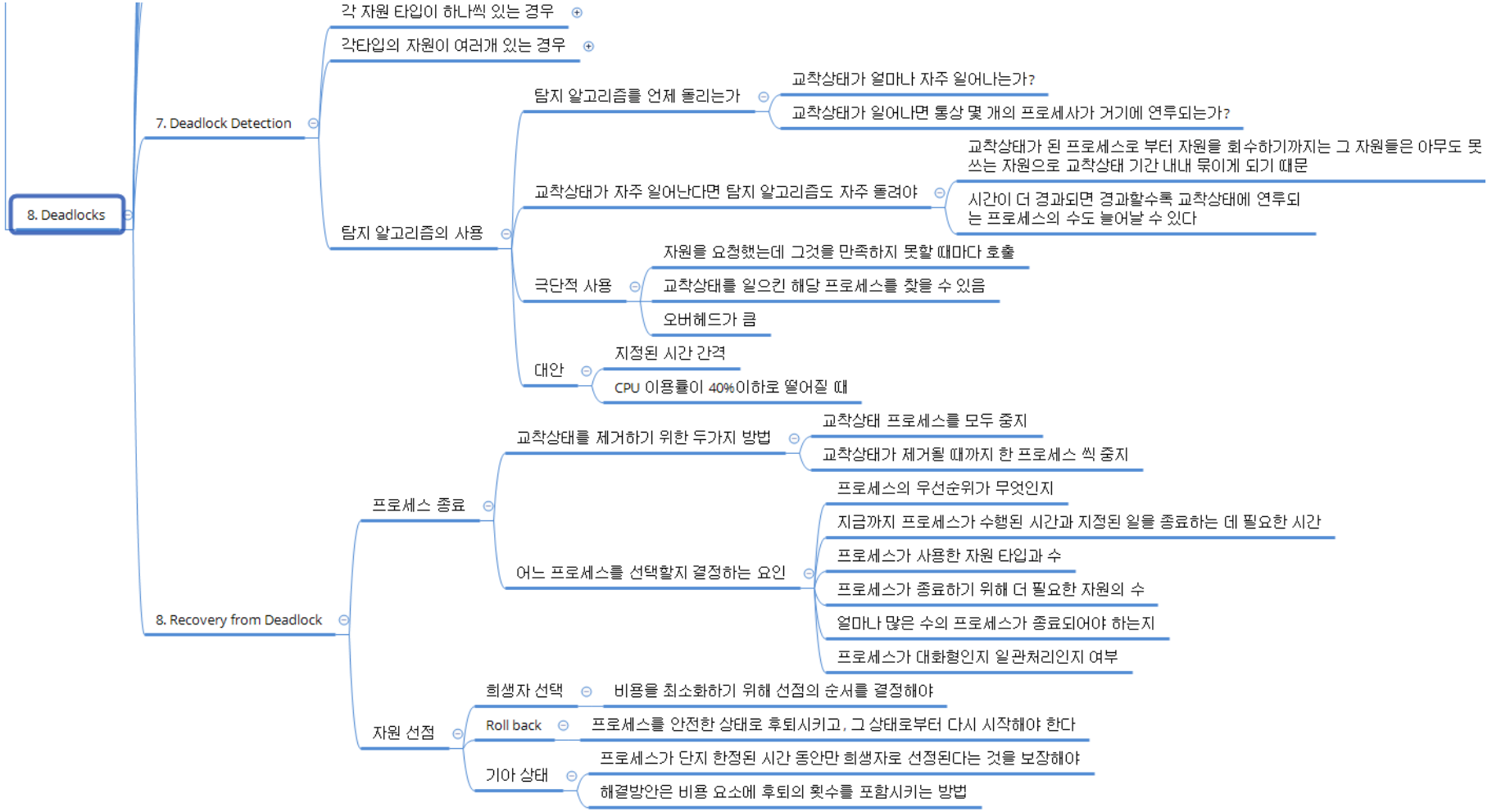
8.8.1 Process Termination

- Abort all deadlocked processes
- Abort one process at a time until the deadlock cycle is eliminated
- In which order should we choose to abort?
  - Priority of the process
  - How long process has computed, and how much longer to completion
  - Resources the process has used
  - Resources process needs to complete
  - How many processes will need to be terminated
  - Is process interactive or batch?

8.8.2 Resource Preemption

- Selecting a victim – minimize cost
- Rollback – return to some safe state, restart process for that state
- Starvation – same process may always be picked as victim, include number of rollback in cost factor

학습한 내용


8. Deadlocks
7. Deadlock Detection
각 자원 타입이 하나씩 있는 경우
각타입의 자원이 여러개 있는 경우
탐지 알고리즘의 사용
탐지 알고리즘를 언제 돌리는가
교착상태가 얼마나 자주 일어나는가?
교착상태가 일어나면 통상 몇 개의 프로세사가 거기에 연루되는가?
교착상태가 자주 일어난다면 탐지 알고리즘도 자주 돌려야
교착상태가 된 프로세스로 부터 자원을 회수하기까지는 그 자원들은 아무도 못 쓰는 자원으로 교착상태 기간 내내 묶이게 되기 때문
시간이 더 경과되면 경과할수록 교착상태에 연루되는 프로세스의 수도 늘어날 수 있다
극단적 사용
자원을 요청했는데 그것을 만족하지 못할 때마다 호출
교착상태를 일으킨 해당 프로세스를 찾을 수 있음
오버헤드가 큼
대안
지정된 시간 간격
CPU 이용률이 40%이하로 떨어질 때
8. Recovery from Deadlock
프로세스 종료
교착상태를 제거하기 위한 두가지 방법
교착상태 프로세스를 모두 중지
교착상태가 제거될 때까지 한 프로세스 씩 중지
어느 프로세스를 선택할지 결정하는 요인
프로세스의 우선순위가 무엇인지
지금까지 프로세스가 수행된 시간과 지정된 일을 종료하는 데 필요한 시간
프로세스가 사용한 자원 타입과 수
프로세스가 종료하기 위해 더 필요한 자원의 수
얼마나 많은 수의 프로세스가 종료되어야 하는지
프로세스가 대화형인지 일관처리인지 여부
자원 선점
희생자 선택
비용을 최소화하기 위해 선점의 순서를 결정해야
Roll back
프로세스를 안전한 상태로 후퇴시키고, 그 상태로부터 다시 시작해야 한다
기아 상태
프로세스가 단지 한정된 시간 동안만 희생자로 선정된다는 것을 보장해야
해결방안은 비용 요소에 후퇴의 횟수를 포함시키는 방법