4장. 교착상태(Deadlock)

4.1 개요

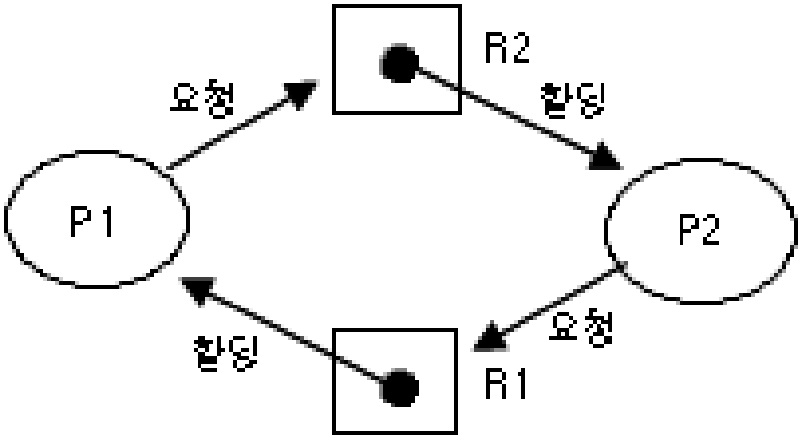
4.2 교착상태 원인

4.3 교착상태 해결책

4.1 개 요

❖교착상태란?

① 다중 프로그래밍 환경에서,
② 서로 다른 프로세스들이 상호간에 점유하고 있는 있는 자원 사용을 요청하고 있으나,
③ 결코 요청한 자원을 영원히 사용할 수 없는 상황.



[그림 4.1] 교착상태

자원할당 그래프(Resource Allocation Graph)

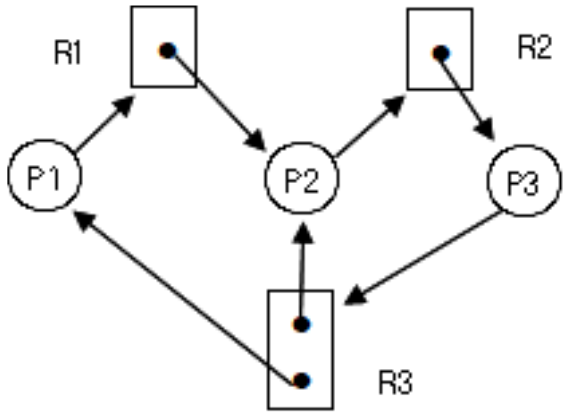
G = (V, E)

V(vertex) : 프로세스와 자원의 집합

E(edge) : 프로세스와 자원의 관계

(사이클-1) : P1 → R1 → P2 → R2 → P3 → R3 → P1

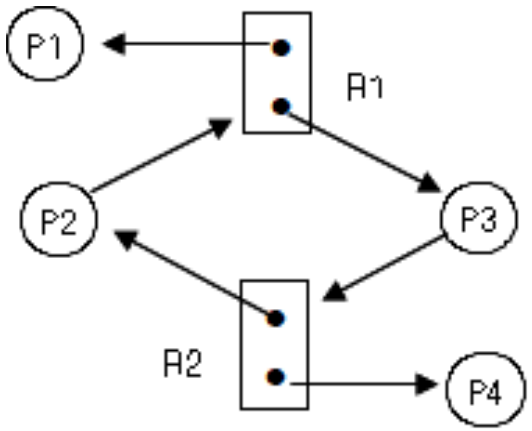
(사이클-2) : P2 → R2 → P3 → R3 → P2



[그림 4.3] 교착상태가 발생한 자원 할당 그래프

자원할당 그래프(Resource Allocation Graph)

(사이클-1) : P2 → R1 → P3 → R2 → P2



[그림 4.4] 사이클이 있지만 교착상태가 아닌 자원 할당 그래프

① 자원할당 그래프를 이용하여 프로세스의 교착상태 여부를 판단할 수 있다.
② 자원할당 그래프에 싸이클(cycle)이 없으면 교착상태의 프로세스가 없다.
③ 자원할당 그래프에 싸이클이 있다고 해서 반드시 교착상태의 프로세스가 존재하는 것은 아니다.

4.2 교착상태 원인

✓ 프로세스가 자원을 사용하고 싶을 땐 자원을 관리하는 운영체제에게 요청한다.

① 요청(request)

② 사용(usage)

③ 해제(release)

❖ 교착상태가 발생하기 위한 필요조건

① 상호 배제(mutual exclusion)

② 점유 및 대기(hold and wait)

③ 비선점(no preemption)

④ 환형대기(circular wait)

4.3 교착상태 해결책

4.3.1 예방책(prevention)

교착상태의 발생 조건(4가지) 중 한가지를 사전에 부정한다.

① 상호배제 조건 부정 : 공유 자원을 두 개 이상의 프로세스가 동시에 접근할 수 있도록 한다.

② 점유 및 대기 조건 부정 : 자원 점유(hold) 혹은 프로세스의 대기(wait) 상태를 부정한다

③ 비선점 조건 부정 : 모든 자원에 대한 선점을 허용한다.

④ 환형 대기 조건 부정 : 자원마다 고유의 번호를 부여하여 오름차 혹은 내림차 순으로만 요청하도록 한다.

✓ 자원 이용률이 매우 저하된다.
✓ 이론적으로 가능하지만 실질적으로 적용하기엔 곤란하다.

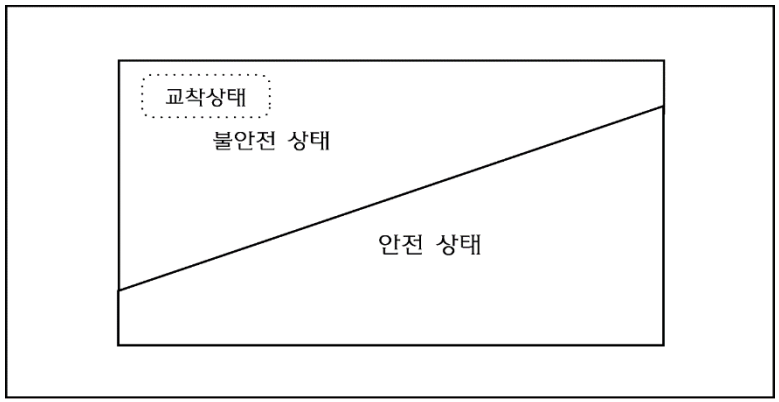
4.3 교착상태 해결책

4.3.2 회피책(avoidance)

❖ 사전에 예방하는 것이 아니고, 교착상태 발생 가능성을 판단하여 자원할당 여부를 결정한다.

※ 교착상태 발생 가능성을 판단하는데 필요한 정보
① 자원 별 사용 가능한 개수
② 각 프로세스가 제시한 자원 별 최대 요구 개수
③ 각 프로세스에게 현재 할당된 자원 별 개수
④ 각 프로세스에게 추가적으로 할당되어야 할 자원 별 개수

✓ 이 정보들을 이용하여 시스템이 항상 안전상태를 유지하도록 자원의 할당 여부를 결정한다.



4.3 교착상태 해결책

4.3.2 회피책(avoidance)

❖ 시스템의 안전상태: 안전 순서(safe sequence)가 존재하면 안전상태이다.

✓ 예를 들어 동일한 유형의 자원이 12개를 가지고 있는 시스템에서 3 개의 프로세스(p1, p2, p3)에 대한 자원 할당 상태가 다음과 같을 경우, 현재 사용 가능한 자원은 3개 이다.

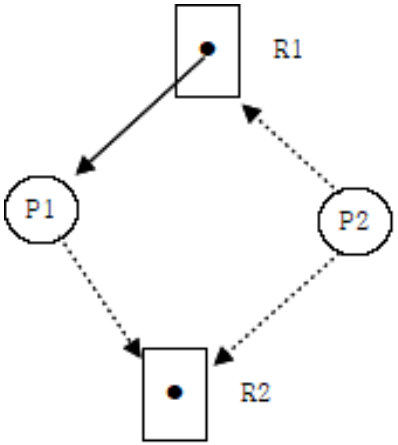
	최대 요구 개수	현재 할당된 개수	추가되어야 할 개수
P1	10	5	5
P2	4	2	2
P3	9	2	7

✓ 안전순서 <p2, p1, p3> 가 존재함으로 현재 시스템의 상태는 안전상태이다.

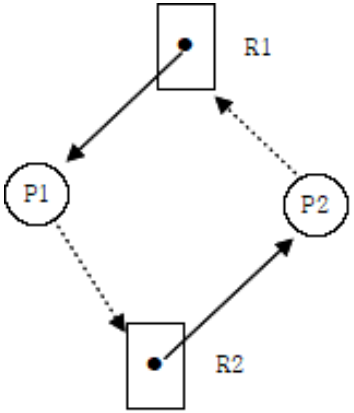
4.3 교착상태 해결책

(2) 회피책(avoidance) 알고리즘

① 자원 할당 그래프를 이용한 알고리즘



(a) 안전 상태



(b) 불안전 상태

4.3 교착상태 해결책

(2) 회피책(avoidance) 알고리즘

② 은행원(Banker's) 알고리즘

자료구조

① Available[n] : 모든 자원(n)에 대한 사용 가능한 개수.
(Available[j]=k 는 j 유형의 자원이 현재 k개 사용 가능함을 의미한다.)

②Max[m,n] : 각각의 프로세스(m)가 제시한 모든 자원(n)에 대한 최대 요구 개수.
(Max[i,j]=k 는 프로세스 i 가 j 유형의 자원을 최대 k개 요구함을 의미한다.)

③Allocation[m,n] : 각각의 프로세스(m)에게 할당된 모든 자원(n)에 대한 개수.
(Allocation[i,j]=k 는 프로세스 i 에게 j 유형의 자원이 k개 할당되었음을 의미한다.)

④Need[m,n] : 각각의 프로세스(m)가 추가적으로 필요로 하는 자원(n)의 개수.
(Need[i,j]=k 는 프로세스 i 가 j 유형의 자원이 k개 추가적으로 필요함을 의미한다.)
(결국, Need[i,j]=Max[i,j]-Allocation[i,j] 임을 유의하라.)

4.3 교착상태 해결책

(2) 회피책(avoidance) 알고리즘

② 은행원(Banker's) 알고리즘

안전(Safty) 알고리즘

①임시 자료 구조 Finish[m], Work[n]의 초기 값을 다음과 같이 선언한다.

Finish[i]:=false (1≤i≤m),

Work:=Available

② 다음의 조건을 동시에 만족하는 프로세스(Pi)를 찾는다.

(조건-1): Finish[i]:= false

(조건-2): $Need_i \leq Work$

만약, 만족하지 않는 프로세스가 하나라도 존재하면 단계 ④로 간다.

③ $Work:=Work+Allocation_i$

Finish[i]:=true

단계 ②로 간다.

④ 모든 프로세스(m)에 대하여 다음의 조건을 만족하는지 조사한다.

(조건): Finish[i]=true (1≤i≤m)

만약, 만족하지 않는 프로세스가 하나라도 존재하면 불안전 상태이다.

4.3 교착상태 해결책

(2) 회피책(avoidance) 알고리즘

② 은행원(Banker's) 알고리즘

알고리즘

① 필요 이상의 자원을 요청하는지 조사한다.

(조건): $Request_i \le Need_i$

만약, 만족하지 않으면 필요 이상의 자원을 요청한 것으로 오류(error)이다.

② 요청한 자원이 할당 가능한지 조사한다.

(조건): $Request_i \le Available$

만약, 만족하지 않으면 자원이 부족함으로 프로세스 P_i를 대기 상태로 처리한다.

③프로세스 P_i가 요청한 자원 할당을 가정하고 다음과 같이 자료 구조를 수정한 후, 안전 알고리즘을 적용하여 안전성 여부를 판단한다.

$Available := Available - Request_i$

$Allocation_i := Allocation_i + Request_i$

$Need_i := Need_i - Request_i$

만약, 불안전 상태이면 프로세스 P_i에게 자원을 할당하지 않고 대기 상태로 처리한 후 자료 구조를 원래대로 복구한다.

4.3 교착상태 해결책

(2) 회피책(avoidance) 알고리즘

③ 은행원(Banker's) 알고리즘 예

자원의 유형(R1, R2, R3) = (9, 5, 7), 프로세스(P1, P2, P3, P4, P5)의 최대 요구와 이미 할당된 상태가 다음과 같다고 가정하자.

	Max	Allocation	Need
	R1 R2 R3	R1 R2 R3	R1 R2 R3
P1	6 5 3	0 1 0	6 4 3
P2	3 2 7	2 0 1	1 2 6
P3	2 1 5	1 0 2	1 1 3
P4	5 2 6	1 1 0	4 1 6
P5	4 3 4	0 0 2	4 3 2

이 경우,

n = 3, m =5,

Available[3] = (9, 5, 7) – (4, 3, 5) = (5, 3, 2) 임으로

안전순서 <P5, P3, P4, P1, P2> 존재한다.

따라서,

현재 상태는 안전상태로 판단한다.

4.3 교착상태 해결책

(2) 회피책(avoidance) 알고리즘

③ 은행원(Banker's) 알고리즘 예

위와 같은 상황에서 임의의 프로세스(P3)가 자원(R1, R2, R3)을 각각 (1, 0, 1)씩 요청하였다고 가정하자. 즉 Request3=(1, 0, 1)을 가정하자.

이 경우, Request3=(1, 0, 1) < Available[3] = (5, 3, 2) 임으로 P3이 요청한 자원 할당을 가정하고 자료 구조를 다음과 같이 수정한 후, 안전 알고리즘으로 안전성을 검사한다.

	Max	Allocation	Need
	R1 R2 R3	R1 R2 R3	R1 R2 R3
P1	6 5 3	0 1 0	6 4 3
P2	3 2 7	2 0 1	1 2 6
P3	2 1 5	2 0 3	0 1 2
P4	5 2 6	1 1 0	4 1 6
P5	4 3 4	0 0 2	4 3 2

이렇게 되면 Available[3] = (4, 3, 1)이 됨으로 안전순서가 존재하지 않는다.

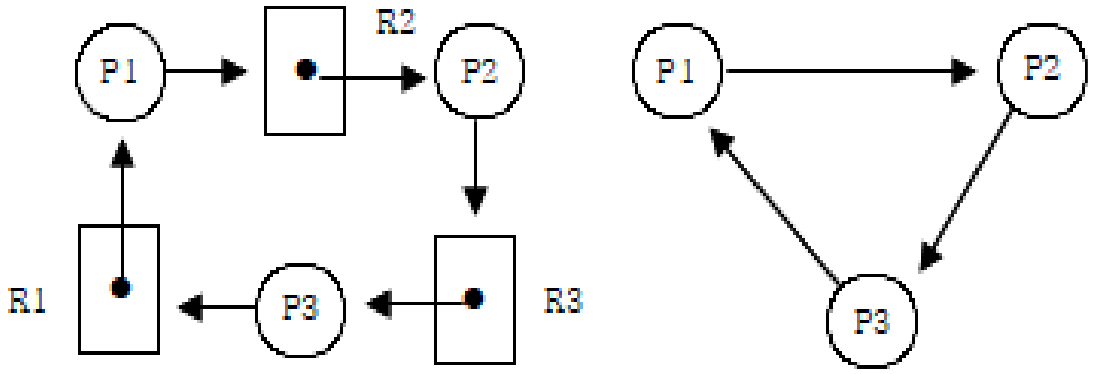
따라서, 불안전상태로 판단하여 P3를 대기상태로 변화시킨다.

4.3 교착상태 해결책

(3) 탐지 및 회복

<탐지 방법>

① 대기 그래프(wait-for graph)를 이용한 탐지



② 교착상태 탐지 알고리즘 이용한 탐지

4.3 교착상태 해결책

(3) 탐지(detection) 및 회복(recovery)

< 회복 방법>

① 프로세스 종료(termination)

- 프로세스의 우선순위
- 지금까지 프로세스가 실행된 시간 및 앞으로 실행해야 할 시간
- 지금까지 프로세스가 사용한 자원 유형 및 개수
- 앞으로 프로세스가 필요로 하는 자원 유형 및 개수
- 다른 프로세스와의 관계

② 자원 선점(preemption)

- 희생자의 선택(selection of a victim)
- 복귀(rollback)
- 기아현상(starvation) :