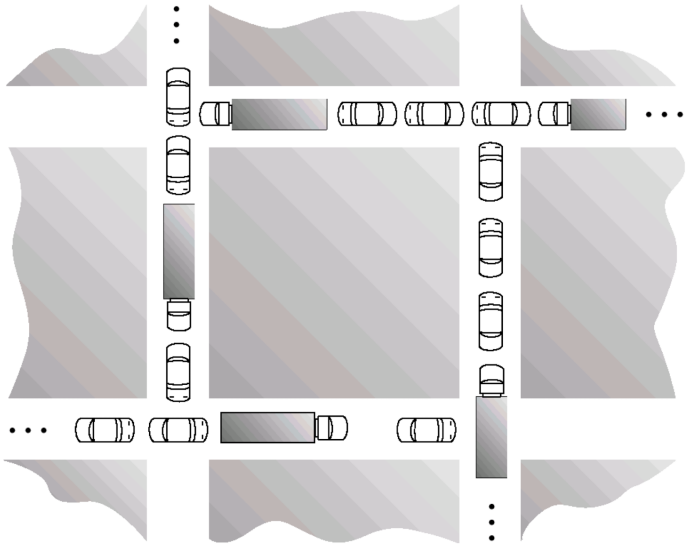
7. Deadlocks

Deadlock이 발생하는 조건 4가지

다음 4가지 조건이 동시에 발생해야 deadlock이 발생할 수 있다(무조건 발생하는게 아님)

1. Mutual Exclusion
   - 하나의 프로세스만 하나의 자원을 점유할 수 있다
2. No preemption
   - 자원의 반환은 자발적으로 이뤄져야 한다(다른 프로세스가 뺐을 수 없음)
3. Hold and Wait
   - 일단 자원 하나를 확보하면 프로세스가 실행될 때까지 이미 잡은 자원은 반환하지 않는다
4. Circular Wait
   - 순환 대기(Cycle이 생성될 때)

현실에서의 Deadlock



Mutual Exclusion: 하나의 도로에 하나의 차만이 존재한다. (차 위에 차가 올라갈 수 없음)

No Preemption: 교차로를 자원이라고 생각할 때 앞에 있는 차가 지나가야만 자원을 반납한다

Hold and Wait: 교차로의 차가 지나갈 때까지 기다린다

Circular Wait: 지금 저 그림을 크게 봤을 때의 상황

Resource-Allocation Graph

deadlock을 판단하기 위해서 프로세스와 자원간의 관계를 볼 예정 ⇒ 이 때 사용하는게 graph

그래프의 구성 ⇒ Vertex, Edge (알고리즘에 있는 그 그래프랑 같음)

Vertex: Process와 Resource로 나뉨 (종류가 2가지)

Edge: request edge(Process→Resource), assignment edge(Resource→Process)

request edge: process가 자원을 요청하는 간선

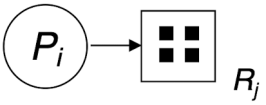
assignment edge: 자원이 process에 할당된 간선

그림으로 나타낸 resource-allocation graph의 구성요소

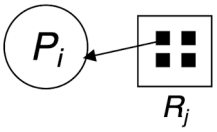
- Process

- Resource Type with 4 instances

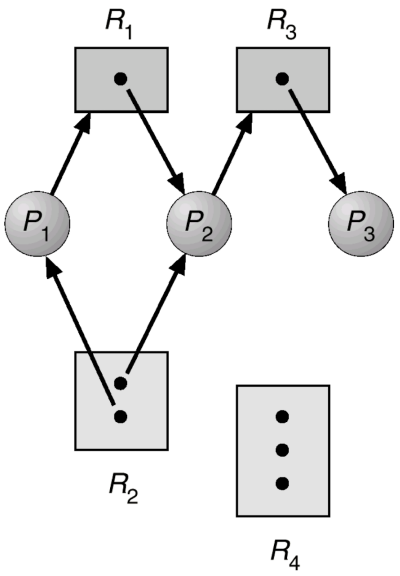
- P_i requests an instance of R_j



- P_i is holding an instance of R_j



이 상황은 데드락일까?



일단 아님

1. P3는 request가 없기 때문에 그대로 실행됨
2. P3가 종료된 이후 R3가 남기 때문에 P2에 R3가 할당됨
3. P2가 필요한 자원은 모두 할당 받았으므로 P2 실행됨
4. P2가 종료된 이후 R1이 P1에 할당됨
5. P1이 필요한 자원을 모두 할당 받았으므로 P1 실행됨

⇒ 데드락 발생 안함

여기서 조건 4가지를 살펴보면

1. Mutual Exclusion
   - R에 들어있는 자원을 하나의 프로세스만이 점유함
2. No Preemption
   - request 요청을 하고 바로 뺐어 버리는게 아니라 기다림
3. Hold and Wait
   - 이미 할당된(assign) 자원들은 프로세스가 점유한 채로 기다림
4. Circular Wait
   - 이게 발생 안함

즉 여기서 살펴볼 수 있는 건 4가지 조건 중 하나라도 만족하지 못하면 데드락이 발생하지 않음

1, 2, 3 조건을 만족할 때

그러면 cycle이 없으면 무조건 데드락이 아닐까?

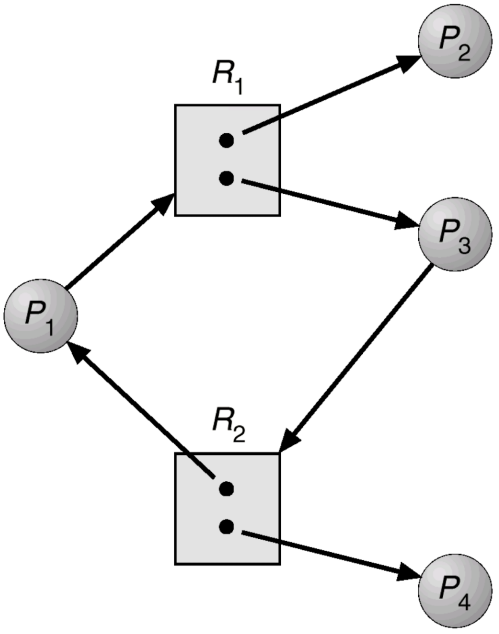
→ ㅇㅇ 무조건 안됨

반대로 cycle이 있으면 무조건 데드락?

→ resource 하나당 instance 하나면 데드락 발생

→ 그게 아니라면 할수도 안할수도 있음

그럼 이건 데드락 발생하나?



ㄴㄴ 발생 안함

사이클이 있지만 데드락이 발생 안함

P4, P2는 알아서 먼저 실행됨

R2와 R1이 풀리면서 P3와 P1도 자원을 잡아서 실행할 수 있게됨

Methods for Handling Deadlocks

1. prevention
2. avoidance
3. detection

prevention, avoidance ⇒ 데드락이 애초에 발생하지 못하도록

detection ⇒ 데드락이 발생하면 그걸 탐지하고 복구하도록

근데 요즘 운영체제?

둘 다 안함, 데드락을 확인하는 게 더 오버헤드가 커서 그냥 프로세스 껐다 켜버리면 해결되니까

Deadlock Prevention

deadlock이 발생하는 조건 4가지 중에서 하나라도 발생하지 못하게 막아버리면 해결

1. Mutual Exclusion

⇒ 이건 막을 수가 없어, 자원 동시에 써버리면 데이터 날아가버리니까

2. Hold and Wait
   - 어떤 자원도 할당 받지 않은 프로세스만이 요청할 수 있도록 하면?
     - 자원 이용률이 낮아지고, starvation 문제가 발생할 수 있어
   - 스레드가 실행하기 전에 모든 자원을 요청하고 할당 받으면?
     - 실행 중에 자원을 추가로 요청해버리면? ⇒ 못 씀

3. No Preemption
   ⇒ 이건 이미 배움, 철학자 문제에서
   자원을 모두 다 확보하려고 하고, 만약 하나라도 확보하지 못한다면 모든 자원을 내려놓으면 됨
   그러면 이거 뭐가 문제냐?
   얘도 자원 이용률이 낮아짐
   (딱 하나만 더 잡으면 되는데 그걸 못잡아서 다 내려놓고 다시 잡아야하니까)
   이거 근데 hold and wait이랑 비슷하지 않나? ⇒ 결과는 같음
   No Preemption은 일단 자원을 잡아보고 안되면 다 방출하는거고
   Hold and Wait은 일단 잡는게 아니라 하나도 안잡은 애들만 잡으려 드는 거

4. Circular Wait
   ⇒ 제일 막기 쉬움, 자원을 사용하는 프로세스들에 순서를 부여해버리면 무조건 막을 수 있어

일단 prevention의 가장 큰 단점은 일어나지 않을 수 있는 일에 대해서 매번 준비를 해야됨
(이걸 확인하는데 자원을 소모하게 돼버리니까)

Deadlock Avoidance

일단 이거도 데드락이 발생하지 않도록 막는 방법인데, prevention이랑 약간 달라
avoidance는 시스템의 상태를 지속적으로 확인해서 안전하지 않은 상태를 회피(자원을 몰아받으면서 처리)
⇒ 이렇게 하면 circular wait이 절대 발생하지 않아서 문제 해결, 그럼 왜 이게 circular wait이 발생하지 않아?

Safe State

어떤 경우에도 데드락이 발생하지 않고 프로세스가 차례대로 실행될 수 있는 상태
모든 자원을 몰아받으면서 프로세스가 순차적으로 실행되면 safe state를 만족

그러면 unsafe state는 데드락이 발생할까?
⇒ 그럴 확률이 생기는거, 만약 특정 프로세스가 끝나버려서 자원이 남아버릴 수 있으니 반드시 발생하는건 아님

Safe State 예시(Banker's Algorithm과 유사)

전체자원: 12

	최대 소요량	현재 사용량
P0	10	5
P1	4	2
P2	9	2

이건 safe state?

⇒ ㅇㅇ P1→P0→P2 sequence가 존재

현재 점유하고 있는 자원이 총 9 ⇒ 남은 자원이 3

일단 P1은 2개만 있으면 실행가능해서 P1 먼저 실행(P0은 5개, P2는 7개가 더 필요해서 불가능)

P1이 모든 자원을 몰아받으면서 실행을 끝내게되면 남은 자원은 5개가 된다

P0가 모든 자원을 몰아받으면서 실행할 수 있게 되고 끝나고 나면 사용 가능한 자원이 10개가 된다

그 다음 P2가 실행

이렇게 자원이 무조건 남는 경우 safe state이고 deadlock이 절대 발생하지 않음

전체자원: 10

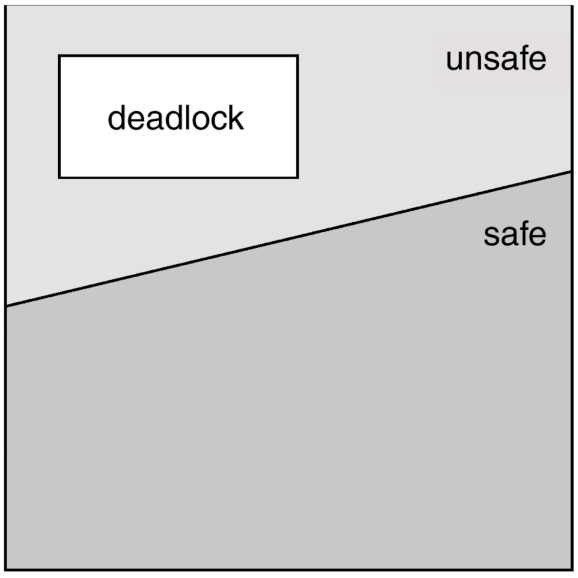
	최대 소요량	현재 사용량
P0	10	5
P1	4	2
P2	9	2

이건 safe state?

⇒ ㄴㄴ

실행할 수 있는 프로세스가 없음

그러면 deadlock이 발생할까? ⇒ 그럴 확률이 있다 정도



그래서 unsafe일 때 항상 발생하는 건 아니고 그럴 수 있다 정도

Avoidance algorithms

1. resource-allocation graph를 이용한 방법
2. banker's algorithm을 이용한 방법

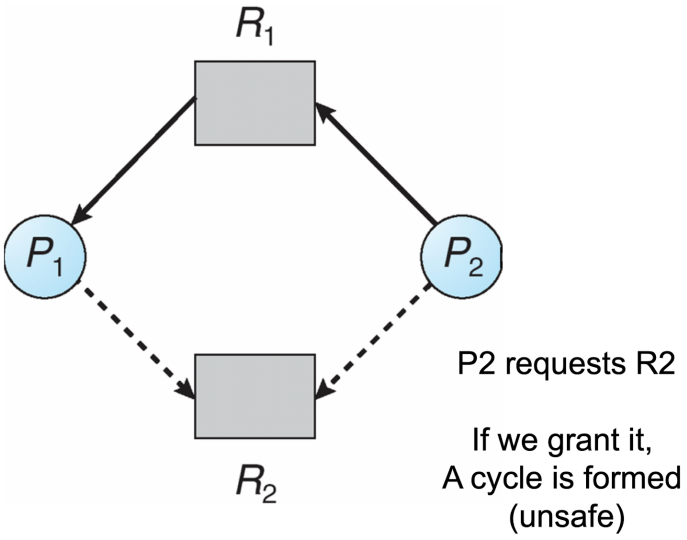
Deadlock Avoidance: Resource-allocation graph

request edge 와 assignment edge는 위에서 이미 봤음

claim edge는 이 프로세스가 실행되려면 필요한 간선인데, 실제로 요청하지는 않은 간선

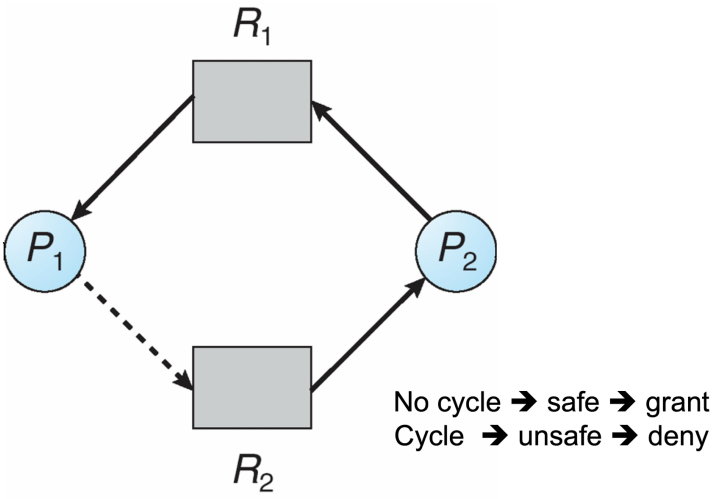
즉 claim edge → request edge → assignment edge 로 넘어감

claim edge(점선), request edge(process→resource), assignment edge(resource→process)



이 상황인데, P1, P2 모두 R2를 필요로 함

이 때 P2가 R2를 request 한 상태가 아래 그림과 같이 나옴



지금 이 그림이 cycle을 구성 ⇒ unsafe

당연히 데드락이 발생하지 않을 수도 있는데, 애초에 이런걸 아예 막아버리는거

Deadlock Avoidance: Banker's Algorithm

모든 프로세스가 자원을 몰아받으면서 순차적으로 실행하는 알고리즘

만약 실행할 수 없다면 대기

이거는 설명보다는 예시로 보는게 이해가 더 빠름

P0~P4로 5개의 프로세스가 존재

Resource type은 A(10), B(5), C(7)로 3종류 존재하고 각각 인스턴스 개수가 다름(괄호안의 숫자)

	Allocation	Max	Available	Need
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 상태에서 safe state 만으로 실행될 수 있는 프로세스의 sequence가 존재?

⇒ ㅇㅇ <P1, P3, P4, P2, P0>

1. 일단 초기상태의 available이 (3 3 2) 라서 P1 or P3 둘 중 하나를 먼저 실행할 수 있음
2. P1실행하고 나면 resource가 (5 3 2) 만큼 남음
3. P3 or P4 실행 가능한데 P3 먼저 실행했다고 치면
4. resource가 (7, 4, 3) 만큼 남음
5. ...

이렇게 쭉 실행하면 결국 모든 프로세스가 safe state를 만족하면서 실행될 수 있음

Deadlock Detection

일단 deadlock은 허용해, deadlock을 식별하면 그 때 대응

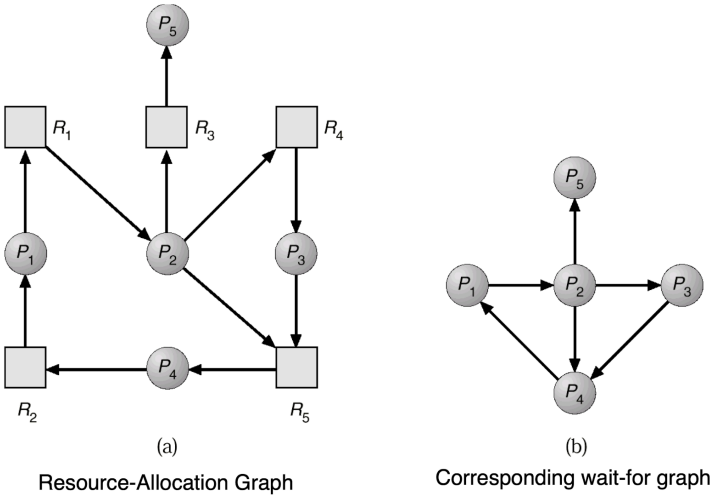
Single Instance of Each Resource Type: Wait-for graph

resource 마다 하나의 instance만 있는 경우 Wait-for graph로 대응가능

Wait-for graph는 프로세스가 어떤 프로세스를 기다리는지 나타내는 그래프

resource allocation graph와는 다르게 wait-for graph는 프로세스간의 관계만 나와있어

(자원과 프로세스와의 관계는 필요 없어서)



그래서 그래프가 저렇게 바뀜

이 때 wait-for graph는 cycle이 생기는 순간 무조건 데드락 발생

Several Instances of a Resource Type

⇒ 이거 위에서 봤던 Deadlock Avoidance랑 다를게 없어

Recovery from Deadlock

Deadlock detection은 데드락이 발생하는 걸 확인하는 거니까 이걸 복구하는 과정이 필요해

1. 데드락을 유발한 프로세스를 종료
2. 데드락과 관련된 모든 프로세스를 종료
3. 자원을 강제로 뺐음(Resource Preemption) ⇒ 애초에 이게 데드락 발생 조건이니까