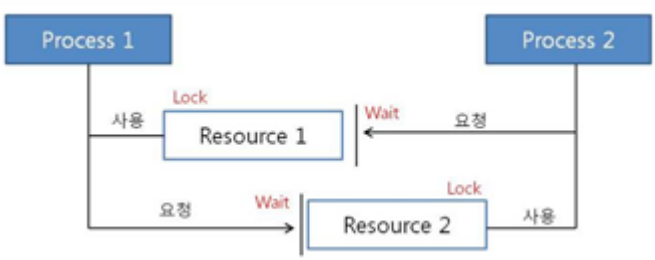
데드락

데드락이란?



- 둘 이상의 프로세스가 다른 프로세스가 점유하고 있는 자원을 서로 기다릴 때 무한 대기에 빠지는 상황
- 시스템적으로 한정된 자원을 여러 곳에서 사용하려고 할 때 발생함
- 위 예시에서 Process 1과 Process2는 모두 자원 1,2를 얻어야 하는 상황임. 그런데, 프로세스 1은 자원 1을 얻고, 프로세스 2는 자원2를 얻은 상황임. 현재 서로 원하는 자원이 상대방에 할당되어 있어, 두 프로세스는 무한정 wait 상태에 빠지게 됨. 이를 Deadlock이라 부름

교착 상태는 왜 일어나는가?

- 프로세스는 실행을 위해 여러 자원을 필요로 함
- 어떤 자원은 갖고 있으나 다른 자원은 갖지 못할 때, 대기를 함
- 한 자원을 가지고 있으면서, 남이 가지고 있는 자원을 기다리다 보면, 자기가 잡고 있는 자원을 안놔주기 때문에, 다른 프로세스 또한 그 자원을 갖지 못하는 경우 교착상태가 발생하기 쉬움

데드락의 발생 조건

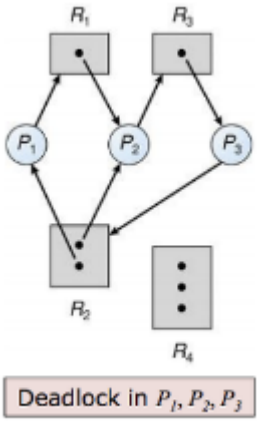
- 4가지 모두 성립해야 데드락이 발생함

교착상태 필요 조건

- **상호 배제(Mutual Exclusion)**

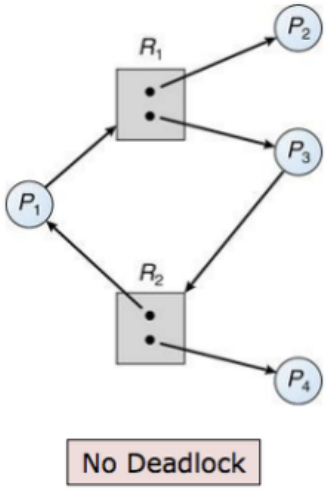
- 자원은 한번에 한 프로세스만 사용할 수 있음
- 자원을 서로 공유하지 못함

- **점유 대기(Hold and Wait)**
  - 최소한 하나의 자원을 점유하고 있으면서 다른 프로세스에 할당되어 사용하고 있는 자원을 추가로 점유하기 위해 대기하는 프로세스가 존재해야 함
  - 어떤 자원은 갖고 있는데, 다른 자원을 갖지 못하여 기다리고 있음
  - 아무것도 갖지 않은 상태에는 교착상태가 일어나지 않음. 어떤 자원을 갖고 있으면서, 또 다른 것을 기다리고 있는 경우에 일어남
- **비선점(No preemption)**
  - 다른 프로세스에 할당된 자원은 사용이 끝날 때까지 강제로 빼앗을 수 없음
  - 다른 프로세스가 가지고 있는 자원을 마냥 기다리기만 한다면 교착상태가 일어남.
  - 강제로 뺏어올 수 있다면, 교착상태가 일어나지 않음 ⇒ But 일반적으로 불가능
- **순환 대기(Circular Wait)**
  - 선형이 아니고 원형을 이루게 되어 프로세스의 자원 할당에서 첫 번째 프로세스와 마지막 프로세스의 자원할당이 겹치게 되어 원형에 있는 모든 프로세스가 자원 할당을 받고자 기다리는 형태가 만들어지는 것을 말한다



  - P1 : R1에 자원을 요청, R2의 자원을 보유
  - P2 : R3에 자원 요청, R1과 R2의 자원을 보유
  - P3 : R2에 자원 요청, R3의 자원 보유

  ⇒ 모든 프로세스는 자원을 가지고 있지만 자원을 요청하고 기다리고 있는 상태



- P2 : R1 사용중, 작업이 끝나면 R1에 대한 자원을 반납가능
- R1이 반남되면, P1이 그 자원을 사용 가능
- P4 : R2의 자원을 사용하고 반납 가능
- P3 : R2의 자원이 만납되면, R2를 사용 가능하여 교착상태가 일어나지 않음

데드락 처리 : 방지, 회피, 검출 및 복구, 무시

교착상태 처리

- 교창상태 **방지(Prevention)**
  - 교착상태의 4가지 필요조건 중 한 가지 이상 불만족하게 하는 것
    - 상호배제 부정 : 자원을 공유 가능하도록 해야함. 하지만, 원천적으로 불가능할 가능성이 높음
    - 점유대기 부정 : 자원을 가지고 있으면서, 다른 자원을 기다리지 않도록 함.
      - 자원이 없는 상태에서 모든 자원 대기 (필요한 자원 한번에 잡기)
      - 일부 자원만 가용하면 모든 자원을 놓아주기

      ⇒ But, 자원의 활용을 저하시킴. ⇒ 기아
    - 비선점 부정
      - 자원 점유 중인 프로세스가 다른 자원을 요구할 때 가진 자원 반납
      - 자원을 선점 가능하게, 일반적으로 불가능함

        - 순환대기 부정
            - 자원에 번호를 부여하고, 항상 자원을 요청할 때에는 자원 번호의 오름차순으로 자원을 요청함. But, 자원의 활용률 저하
            - (자원을 순환 형태로 대기하지 않도록 일정한 한 쪽 방향으로만 자원을 요구할 수 있도록 함)

- 교착상태 **회피(Avoidance)**
    - 교착상태 자체를 자원 요청에 대해 os에서의 잘못된 승인이 이루어져 발생한 것으로 생각함

    ⇒ 운영체제는자원을 할당할 때 불안전 할당이 되지 않도록 해야한다.

    - 프로세스가 자원을 요구할 때, 시스템은 자원을 할당한 후에도 안정 상태로 남아있게 되는지 사전에 검사하여 교착 상태 회피
    - 안정 상태면 자원 할당, 아니면 다른 프로세스들이 자원 해지까지 대기

은행원 알고리즘

- 어떤 자원의 할당을 허용하는지에 관한 여부를 결정하기 이전에, 미리 결정된 모든 자원들의 최대 가능한 할당량을 가지고 시뮬레이션하여 안전 상태에 들 수 있는지 여부를 검사하는 알고리즘


```
40원, 50원, 60원을 필요로 하는 사람 3명
- 은행은 각각 25원씩 주기로 결정함
   => 각각의 사람들은 15, 25, 35원이 더 필요한 상태 // 은행은 25원이 남은 상태

해결 방법?
```

```
1) 15원이 더 필요한 첫번째 고객에게 15원을 빌려줌. 첫번째 고객이 일을 해결하고
갚을 때까지 기다리고 이후 다른 고객에게 돈을 빌려줌
=> 안전
2) 두번째 고객에게 25원을 빌려주고, 두번째 고객이 갚을 때까지 기다림.
=> 안전

BUT,
세번째 고객이 급하다 하여, 25원이 아닌 45원을 달라고 한 경우
(첫번째 고객 25, 두번째 25, 세번째 45원을 주어 5원만 남은 상태)
=> 은행은 5원만 남아 있기에, 세 고객 중 누구도 해결해줄 수 없어, 고객이 일을 해결
하고 갚을 일도 없음. => 불안전상태 (데드락 상태)
```

- 교착상태 **탐지 & 회복**
  - 위 두가지 방법은, 애초에 교착상태가 일어나지 않게 하는 방법이었다면, 이는 교착상태는 일어나도록 허용하되, 발생하면 복구하도록 하는 방법임
  - *탐지 기법*
    - 데드락이 발생했는지 여부를 탐색함.
  - *회복 기법*
    - 데드락을 탐지 깁버을 통해 발견했다면, 순환 대기에서 벗어나 데드락으로부터 회복하기 위한 방법을 사용함
      - 프로세스 종료 방법
        - 교착 상태에 빠진 모든 프로세스를 중단 ⇒ 부작용 발생 가능
        - 교착 상태가 제거될 떄까지 하나씩 프로세스 중지
          - 프로세스를 하나씩 중단 시킬 때마다 탐지 알고리즘으로 데드락을 탐지하면서 회복

            ⇒ 매번 탐지 알고리즘 호출, 수행 ⇒ 부담
      - 자원 선점하기
        - 프로세스에 할당된 자원을 선점해서(뺏어서), 교착 상태를 해결할 때까지 그 자원을 다른 프로세스에 할당
        - 우선 순위가 낮은 프로세스나 수행 횟수 적은 프로세스 위주로 프로세스 자원 선점

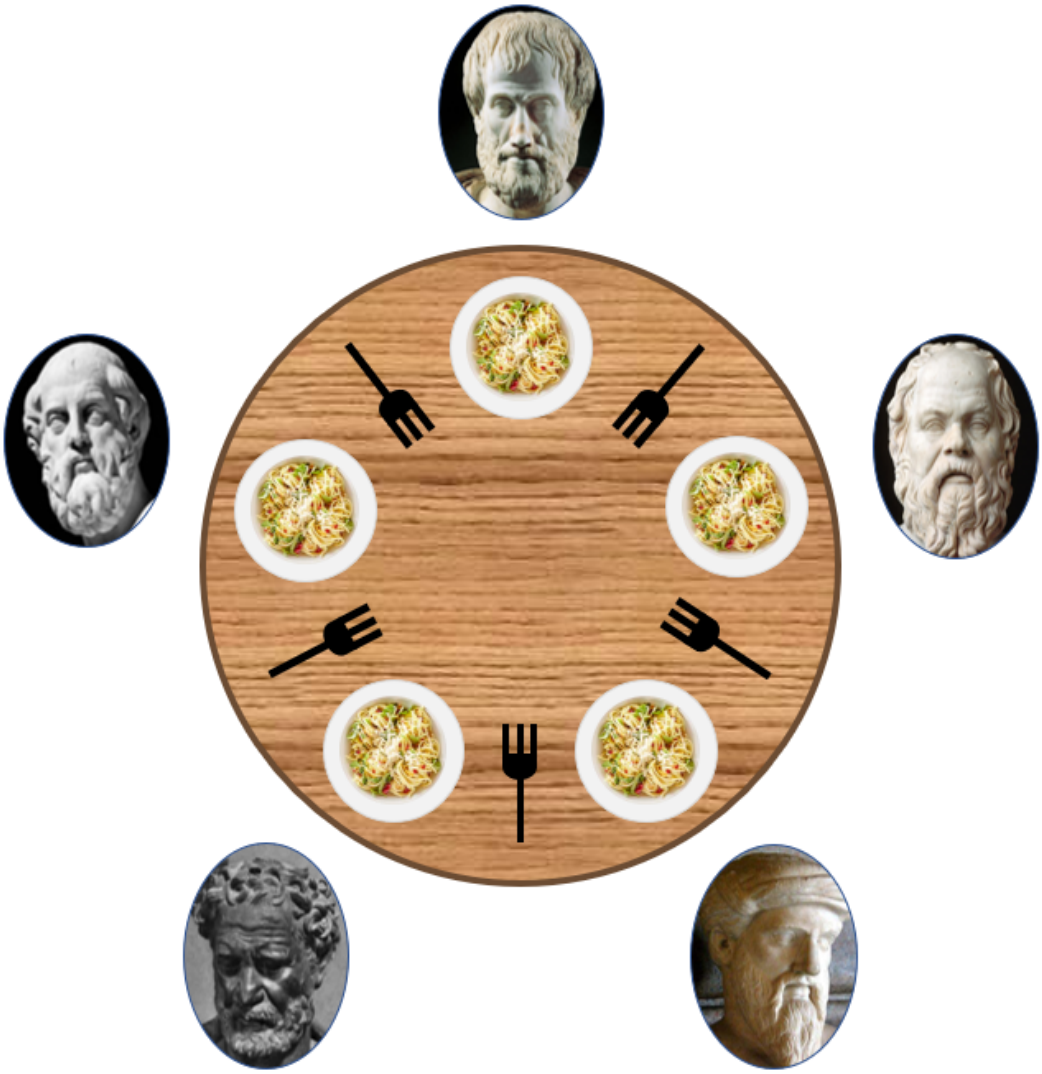
- 교착상태 **무시**

- 교착상태 자체를 무시하는 것
- 실제로 교착상태가 자주 일어나기 힘들고, PC는 일반적으로 혼자 쓰므로 아예 무시함

나올만한 질문

1. 데드락이란?
2. 데드락의 발생 조건은?
3. 은행원 알고리즘이란?
4. 식사하는 철학자?

▼ 문제



> 1. 일정 시간 생각을 함.
> 2. 왼쪽 포크가 사용 가능할때까지 대기함, 사용 가능하면 집어든다
> 3. 오른쪽 포크가 사용 가능해질 때까지 대기한다. 만약 사용 가능하다면 집어든다.
> 4. 양쪽의 포크를 잡으면 일정 시간만큼 식사를 한다.
> 5. 오른쪽 포크를 내려놓는다.
> 6. 왼쪽 포크를 내려놓는다.
> 7. 다시 1번으로 돌아간다.

⇒ 모든 철학자들이 왼쪽 포크를 잡으면, 모든 철학자들은 오른쪽 포크가 사용 가능해질때까지 기다려야함.

=⇒ 모든 철학자가 3번 상태에 머물러, 아무것도 진행할 수 없고 이를 교착 상태라 함

⇒ 아무도 먹지 못하여 기아현상으로 굶으죽게 됨.

5. 데드락 해결방법 4가지
6. 교착상태와 starvation의 차이는?

   교착상태는 프로세스의 상태 중 Waiting(block) 상태일 때 발생하며, 절대 발생하지 않을 이벤트를 무한히 대기한다. 필요한 자원이 공유자원이다. Starvation은 프로세스의 ready 상태일 때 발생하며, 높은 우선순위를 가진 프로세스에 의해 원하는 자원을 계속 선점당하여 무한히 대기하는 것이다. 필요한 자원은 CPU 할당이다.