13.교착_상태

13.교착_상태

13.1. Introduce

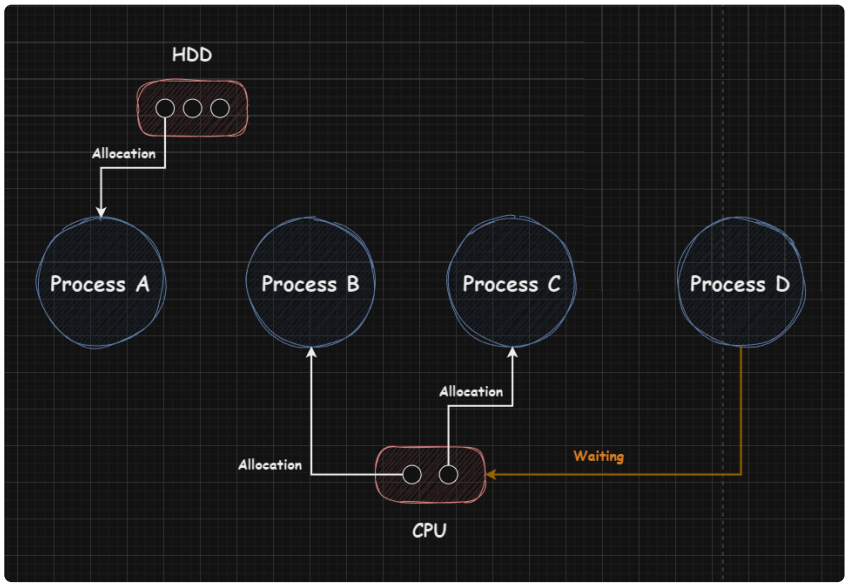
13.1.1. 식사하는 철학자 문제

- Dining philosophers problem
- 동그란 원탁에 다섯 명의 철학자가 앉아 식사를 한다. 이때 철학자 사이에는 포크가 하나씩 있고, 음식은 두 개의 포크를 사용해야 한다.
- 철학자들은 아래와 같은 순서로 식사한다.
  1. 생각하다 왼쪽 포크가 사용 가능하면 집어든다.
  2. 생각하다 오른쪽 포크가 사용 가능하면 집어든다.
  3. 양쪽 포크를 모두 집어들면 정해진 시간동안 식사한다.
  4. 오른 포크, 왼쪽 포크 순으로 내려놓는다.
  5. `1.`을 반복한다.
- 이때 모든 철학자가 동시에 포크를 집어들 경우, 다른 철학자들이 포크 를 내려놓을 때까지 기다려야 하며, 이처럼 이러나지 않을 사건을 기다리며 멈춰버리는 현상을 교착 상태(deadlock)라 한다.
- 여기서 철학자, 포크, 생각하는 행위는 각각 프로세스(스레드), 자원, 그리고 자원을 사용하기 위한 대기 상태에 빗댈 수 있다.
- 또한, 포크는 한 번에 하나의 프로세스(스레드)만 접근할 수 있다는 점에서 **임계 구역**이라 할 수 있다.

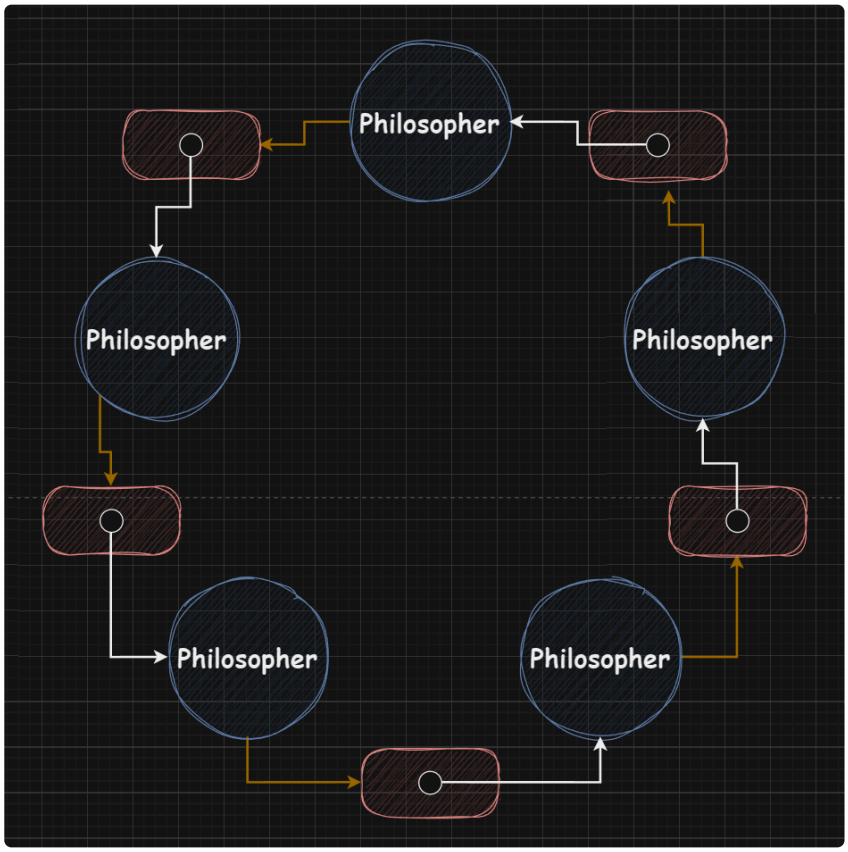
```
// process A
lock1 = true;
while (lock2 == true)
;
// 임계 구역 작업
lock1 = false;

// process B
lock2 = true;
while (lock1 == true)
;
// 임계 구역 작업
lock2 = false;
```

- 위 예처럼 프로세스 A는 임계 구역 진입 전 `lock1`을 잠그고, 프로세스 B는 임계 구역 진입 전 `lock2`를 잠군다.
- 만약, 프로세스 A/B가 각각 서로 다른 프로세스의 `lock`이 `false` 되길 기다린다면 교착 상태가 발생한다.
- 이를 해결하기 위해서는
  - 교착 상태가 발생할 때의 상황을 정확히 표현하고
  - 발생의 근본적인 이유를 알아야 한다.

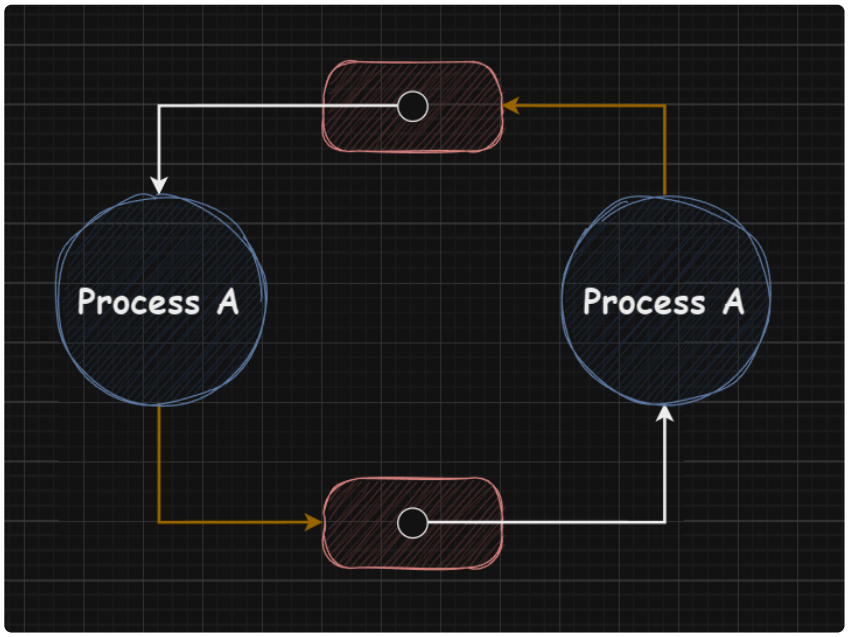
13.1.2. 자원 할당 그래프



- Resource-allocation graph
- 어떤 프로세스가 어떤 자원을 사용하는지, 어떤 자원을 기다리는지 표현하는 간단한 그래프
- 다음 규칙을 따라 그릴 수 있다.
  1. 프로세스는 원으로, 자원의 종류는 사각형으로 그린다.
  2. 자원의 개수는 자원 사각형 내의 점으로 표현한다.
     - 예로, 자원의 종류(CPU, HDD)는 하나라도 자원의 개수는 여럿일 수 있다.
  3. 자원을 할당받아 사용중이라면 자원에서 프로세스를 향해 화살표로 표시한다.
  4. 프로세스가 대기상태라면 프로세스에서 자원으로 화살표를 그린다.



- 철학자 문제도 위 처럼 그래프로 표현(철학자는 프로세스, 포크는 자원) 가능하다.



- 위 그림 처럼 교착상태는 자원 할당 그래프가 원의 형태를 띈다.

13.1.3. 교착 상태 발생 조건

- 상호 배제 Mutual exclusion
  - 교착 상태의 근본적 원인은 해당 자원을 한 번에 하나의 프로세스만 이용 가능하기 때문이다.
- 점유와 대기 Hold and wait
  - 자원을 보유한 채 다른 자원을 기다리는 경우 문제가 발생한다.
- 비선점 nonpreemptive
  - 프로세스의 작업이 끝나야지만 자원을 할당받을 수 있는 등, 다른 프로세스가 자원을 강제로 빼앗을 수 없는 것도 문제가 된다.
- 원형 대기 circular wait
  - 프로세스들과 프로세스가 요청 및 할당받은 자원이 원의 형태를 띌 때 발생할 수 있다.
  - 단, 반드시 발생하는 것은 아니다.

13.2. 교착 상태 해결 방법

13.2.1. 교착 상태 예방

- 교착 발생 조건 중 하나를 충족하지 못하게 만드는 방법.
- 상호 배제 예방:
  - 모든 자원을 공유할 수 있게 함. 단, 이 방법은 현실적으로 어려움.
- 점유와 대기 예방:
  - 프로세스는 자원 하나를 점유한 채로 다른 자원을 위해 대기할 수 없으며, 필요한 모든 자원을 할당 받거나 아예 할당받을 수 없다.
  - 이처럼, 프로세스 하나에 자원을 몰아주는 방식은 당장 자원이 필요하지만 대기하는 프로세스, 사용되지 않으면서 오래 할당되는 자원을 다수 양산하기에 **자원의 활용률이 낮아진다**.
  - 많은 자원을 사용해야 하는 프로세스에게 불리하다. 즉, 기아 현상을 야기할 수 있다.
- 비선점 조건 예방:
  - CPU와 같이 선점할 수 있는 자원에 대해서는 효과적이다.
  - 단, 프린터 같이 한 번에 하나의 프로세스만 사용 가능한 경우 처럼 모든 자원이 선점 가능한 게 아니며, 범용성이 떨어지는 방법이다.
- 원형 대기 조건 예방:
  - 모든 자원에 번호를 붙이고 오름차순으로 자원을 할당해 원형 대기를 예방할 수 있다.
  - 비교적 현실적이로 실용적이지만, 모든 컴퓨터 시스템 내 자원에 번호를 붙이는 작업의 어려움, 붙이는 번호에 따른 활용률의 차이 등이 발생할 수 있다.

13.2.2. 교착 상태 회피

- 교착 상태는 한정된 자원의 무분별한 할당이 원인이라 간주하며, 배분할 수 있는 자원의 양을 고려하여 교착 상태가 발생하지 않을 정도로만 자원을 배분하는 방법.
- 안전 상태 safe state
  - 교착 상태가 발생하지 않고 모든 프로세스가 정상적으로 자원을 할당받아 종료될 수 있는 상태
  - 안전 순서열에 따라 자원을 배분, 교착 상태가 발생하지 않음.
- 불안전 상태 unsafe state
  - 교착 상태가 발생할 수 있는 상황
  - 안전 순서열이 없는 상태
- 안전 순서열 safe sequence
  - 교착 없이 안전하게 프로세스들에게 자원을 할당할 수 있는 순서

프로세스	요구량	현재 사용량
P1	10	5
P2	4	2
P3	9	2

- 위처럼 프로세스는 자원의 최대 사용량을 미리 선언해야 한다.
- OS가 할당 가능한 자원을 총 12이며, 현재 할당한 자원은 9, 남은 자원은 3인 상황이다.
- 만약, 각 프로세스가 최대로 자원을 요구하더라도, 이 상태는 안전 상태로 P2 → P1 → P3 안전 순서열이 존재한다.

프로세스	요구량	현재 사용량
P1	10	5
P2	4	2
P3	9	3

- 만약 위의 표 처럼 자원이 할당되어 있고 남은 자원이 2라면 P2를 종료하고 자원을 반환 받아도, 다음에 종료시킬 수 있는 프로세스가 없다.
- P1, P3는 서로가 보유한 자원을 무한정 기다릴 수 밖에 없는 불안정 교착 상태가 된다.
- 이같은 경우를 피하기 위해 OS는 안전 상태에서 안전 상태로 이동할 수 있는 경우, 즉 자원의 할당이 안전한 경우에만 자원을 할당한다.

13.2.3. 교착 상태 검출 후 회복

- 교착 상태 발생을 인정 후 사후에 조치하는 방식
- OS는 자원 요그에 따라 즉시 자원을 할당하며, 교착 상태 발생 여부를 주기적으로 검사한다.
- 교착 상태가 발생했다면 다음과 같이 회복한다.
  - 선점을 통한 회복
    - 교착 상태가 해결될 때까지 하나의 프로세스씩 자원을 몰아주는 방식
    - 다른 프로세스의 자원을 강제로 빼앗아 할당한다.
  - 강제 종료를 통한 회복
    - 가장 단순하고 확실한 방법
    - 교착 상태가 없어질 때까지 프로세스를 모두 혹은 하나씩 강제 종료한다.
    - 모두 종료하는 경우 프로세스들의 작업 내역을 잃을 수 있고, 하나씩 종료하는 경우 교착 상태를 확인하는 과정에서 오버헤드를 야기한다.
- 이외에 데드락을 시스템이 책임지지 않는 타조 알고리즘 ostrich algorithm을 채용하기도 한다.