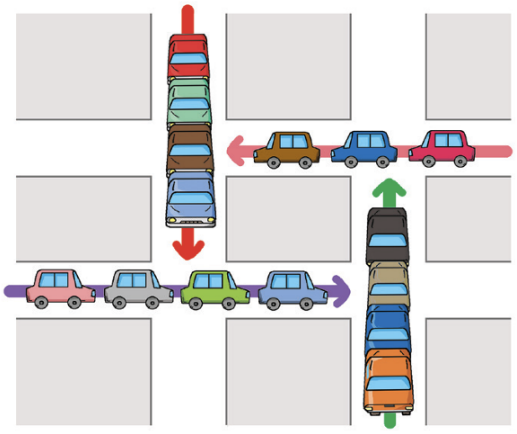
데드락

Deadlock

실생활의 교착상태

한 사람이 밥을 먹기 위해서는 숟가락, 젓가락이 모두 필요한 상황.
A는 숟가락을 들고, B는 젓가락을 들고,
A는 젓가락을 사용할 수 있을 때까지 대기,
B는 숟가락을 사용할 수 있을 때까지 대기,
무한 대기 발생



자동차들이 한 길을 점유하고
다른 길을 막고 있는 경우

컴퓨터 시스템에서의 교착상태

프로세스들이 서로 가진 자원을 기다리며 BLOCK된 상태

→ 각 스레드가 다른 스레드가 소유한 자원을 요청하며 무한정 대기하는 현상

- 자원
  - 하드웨어, 소프트웨어 등을 포함하는 개념
  - 프로세스가 자원을 사용하는 절차
    - request (요청)
    - allocate (할당)

    - use (사용)
    - release (해제)

💡 교착상태는 어디서 발생하는가?

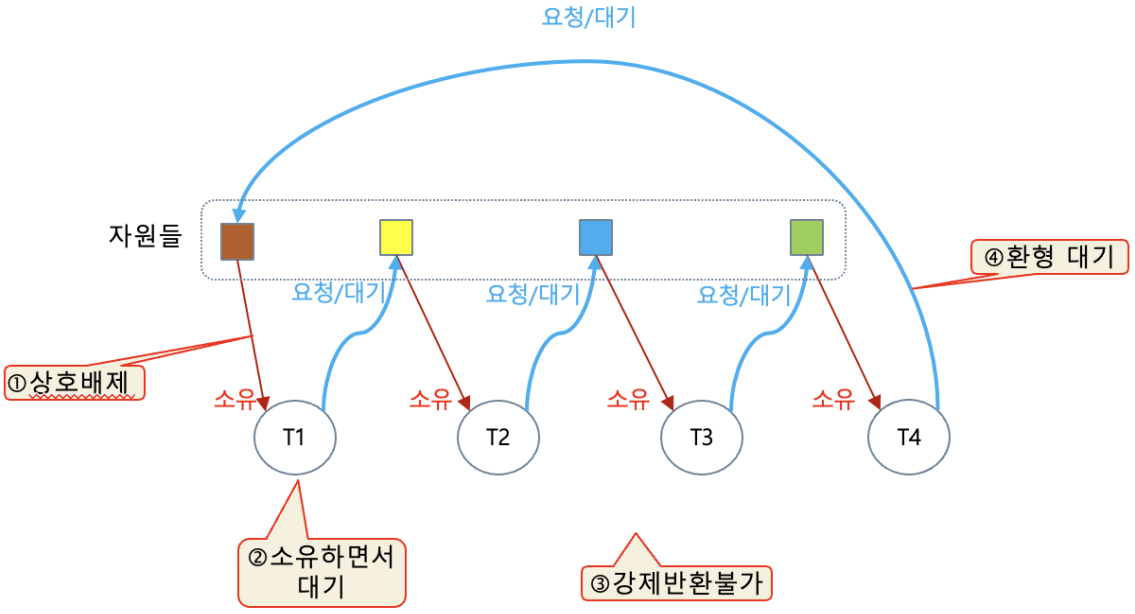
멀티스레드 응용프로그램에서 주로 발생

커널 내에서도 발생함 → 정교하게 작성되어있기에 거의 발생하지 않는다.

교착상태를 막도록 하는 방법은 있는가?
- 막는데 많은 시간적 공간적 비용이 들기에 완벽하게 막을 수는 없지만 예방 및 회피 무시등의 방법을 적용한다.

데드락 발생 조건 : 코프만 조건

- mutual exclusion (상호배제)
    - 공유 자원에 대해 락을 걸어 다른 프로세스가 접근하지 못하게 함
- no preemption (비선점)
    - 다른 프로세스가 서로의 자원을 빼앗을 수 없는 상황
- hold and wait (점유와 대기)
    - 공유 자원을 점유한 상태에서 다른 공유 자원을 사용하기 위해 대기하고 있는 상황
- circular wait (환영 대기)
    - 순환, 원형(사이클) 형태로 서로 필요한 자원을 점유하고 대기하는 상황



데드락 발생

데드락 모델링

Resource Allocation graph (자원 할당 그래프)

- 자원에 대한 시스템의 상태를 나타내는 방향성 그래프
- 자원 할당 그래프를 통해 교착상태 판단

⇒ *교착상태 예방, 회피, 감지를 위한 알고리즘 개발에 필요*

데드락 처리 방법

- deadlock prevention
  - 발생조건 중 하나의 조건만 끊어주면 됨
- deadlock avoidance
  - 데드락의 가능성이 없는 경우에만 자원을 할당
- deadlock detction and recovery
  - deadock 발생은 허용하지만 발생 시 recover함
- deadlock ignorance
  - 데드락을 시스템이 책임지는 것이 아님

deadlock prevention

교착상태 발생 조건 중 최소 하나를 성립하지 못하게 함

- 상호배제 → 상호배제 없애기
  - 동시에 2개 이상의 스레드가 자원을 활용할 수 있도록 함
  - *문제점 : 동시에 공유해서 안되는 자원의 경우 반드시 성립해야하기에 완전히 조건을 방지하는 것은 불가능*
- 점유와 대기 → 기다리지 않게 함
  - 프로세스가 자원을 요청할 때 다른 어떤 자원도 가지고 있지 않게 함
  - 운영체제는 시작시에 모든 필요한 자원을 파악하고 실행 시에 한번에 할당받게 하는 방법
  - 자원을 요청 시에 보유하고있던 모든 자원을 놓고 다시 요청
  - *문제점 : 불가능 혹은 비효율적 → 그렇게 하는게 더 나았으면 진작 되었을듯*
- 비선점 → 선점 허용
  - 모든 프로세스가 자원을 얻을 수 있을 때 프로세스는 다시 시작됨
  - *문제점 : 오버 헤드가 크다*
- 환형 → 환형 대기 제거
  - 자원의 할당 순서를 정해서 정해진 순서대로만 자원 할당

deadlock avoidance

미래에 교착상태가 될 것 같으면 자원을 할당하지 않는 것

자원요청에 대해 부가정보를 이용해 자원할당이 데드락으로 부터 안전한지 동적으로 조사하고 안전한 경우에만 할당

safe state : 시스템 내의 프로세스들에 대한 safe sequence(요청하는 자원을 순서대로 실행가능한)가 존재하는 상태

unsafe state : 환형대기에 빠지는 상황

- 시스템이 unsafe state에 들어가지 않는 것을 보장한다.
    - single instance : resource allocation graph algorithm 사용
    - multiple instance : 뱅커 알고리즘 사용
- resource allocation graph algorithm (자원 할당 그래프 알고리즘)
    - cycle이 생기지 않는 경우에만 요청 자원 할당
    - cycle생성 조사하는데에는 O(n^2)의 시간복잡도 발생
- 뱅커 알고리즘
    - 자원 할당 전에 교착상태가 발생하지 않을 것인지 안전한지 판단하는 알고리즘
    - 테이블을 만들어 자원을 할당할지 안할지 결정
    - 판단
        - 각 프로세스가 실행 시작 전에 필요한 전체 자원의 수를 운영체제에게 알린다.
        - 자원을 할당할 때마다 자원을 할당해주었을 때 교착상태가 발생하지 않을 만큼 안전한 상태인지 판단해서 안전한 상태일 때만 자원 할당
        - 필요한 자원의 개수, 프로세스가 할당 받은 자원의 개수, 할당 가능한 자원의 개수를 토대로 현재 요청된 자원을 할당해도 안전한지 판단한다.
    - *문제점 : 비현실적이다. , 프로세스의 개수가 동적으로 변하기에 미리 프로세스의 개수를 정적으로 고정시키는 것은 불가능*

deadlock detection and recovery

- deadlock detection
    - single instance 인 경우
        - cycle이 곧 deadlock
    - multiple instance
        - 뱅커 알고리즘과 유사한 방법 활용
- recovery
    - process termination, kill process
        - 모든 프로세스 죽임
        - 죽이면서 데드락 사라지는지 확인

    - resouce preemption
        - 교착상태에 빠진 비용 최소화할 victim 선점하고 자원을 빼앗아 다른 프로세스에게 할당
    - rollback
        - 교착상태가 발생할 것으로 예측되는 스레드의 상태를 저장하고, 교착상태가 발생하면 마지막으로 저장된 상태로 돌아가게 하여 다시 시작하면서 자원을 다르게 할당
        - 문제점
            - 계속같은 프로세스가 victim으로 선정되는 경우
            - rollback 횟수도 같이 고려해야함
- *문제점 : 데드락을 감지하는 프로세스가 늘 실행되어야 하는 부담*

deadlock ignorance

- 거의 일어나지 않을 교착상태를 해결책을 세워야하나? 에서 시작됨
- deadlock을 무시하고 아무런 조치를 취하지 않음
    - deadlock 처리에 대한 더 큰 오버헤드를 방지하는 차원
    - 모든 데드락을 무시하는 것이 아닌 시스템에 어떤 파국을 부르지 않는 작업들에 대해서 교착상태를 무시하는 것 → Ostrich 알고리즘
        - 교착상태의 발생 가능성은 극히 적다. → 교착상태에 대한 통계치는 없음
        - 발생하지 않을거다 ~ 하고 눈가리고 아웅하는 방식
    - 사용자가 직접 프로세스를 죽이는 방식
    - UNIX, Windows 등 대부분 OS
- 그렇다면 어떻게?
    - 교착상태 예방, 회피, 감지에는 많은 오버헤드가 생기므로 교착상태가 발생하면 시스템 재시작 혹은 특정 프로세스나 스레드를 강제 종료하는 방식
        - 물론 관련 데이터 손실에 대한 문제는 있지만
        - 전체적으로 크지 않을거다라고 생각함