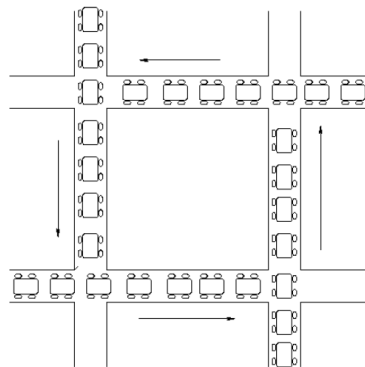




Ch08 : Deadlocks

Deadlock

- 다중 프로그래밍 환경에서는 여러 스레드가 한정된 자원을 사용하려고 서로 경쟁할 수 있다. 한 스레드가 자원을 요청했을 때, 그 시각에 그 자원을 사용할 수 없는 상황이 발생할 수 있고, 그때는 스레드가 대기 상태로 들어간다.
- 이처럼 대기 중인 스레드들이(그들이 요청한 자원들이 다른 스레드들에 의해서 점유되어 있고 그들도 다 대기 상태에 있기 때문에) 결코 다시는 그 상태를 변경시킬 수 없으면 이런 상황을 **교착 상태**라 부른다.



■ Let S and Q be two semaphores initialized to 1

P_0	P_1	
P(S);	P(Q);	하나씩 차지
P(Q);	P(S);	상대방 것을 요구
\vdots	\vdots	
V(S);	V(Q);	여기ওয়া야 release 함
V(Q);	V(S);	

Deadlock

- Deadlock
 - 둘 이상의 프로세스들이 각각 다른 프로세스들 중 하나가 무엇인가를 하기를 기다리고 있기 때문에 진행할 수 없는 상황.

- Livelock
 - 두 개 이상의 프로세스가 유용한 작업을 하지 않고 다른 프로세스(들)의 변화에 대응하여 지속적으로 상태를 변경하는 상황.
- Starvation
 - 실행 가능한 프로세스가 스케줄러에 의해 무한정 간과되는 상황; 진행할 수는 있지만 결코 선택되지 않는 상황.

시스템 모델

- 시스템은 경쟁하는 스레드들 사이에 분배되어야 할 유한한 수의 자원들로 구성된다. 이들 자원은 다수의 유형으로(또는 클래스로) 분할되며, 이들 각각은 동등한 다수의 인스턴스(instance)들로 구성된다. CPU 주기, 파일, 그리고 입/출력 장치(네트워크 인터페이스, DVD 드라이브 등과 같은) 등이 자원 유형들의 예이다.
- 스레드는 자원을 사용하기 전에 반드시 요청해야 하고, 사용 후에는 반드시 방출해야 한다. 스레드는 지정된 태스크(task)를 수행하기 위해 필요한 만큼의 자원을 요청할 수 있다. 명백히 요청된 자원의 수는 시스템에서 사용 가능한 전체 자원의 수를 초과해서는 안 된다.
- 정상적인 작동 모드 하에서, 프로세스는 다음 순서로만 자원을 사용할 수 있다.
 1. **요청** : 스레드는 자원을 요청한다. 요청이 즉시 허용되지 않으면(예를 들어, mutex 락을 다른 스레드가 가지고 있는 경우), 요청 스레드는 자원을 얻을 때까지 대기해야 한다.
 2. **사용** : 스레드는 자원에 대해 작업을 수행할 수 있다. (예를 들어, 자원이 mutex 락이라면, 스레드는 자신의 임계구역에 접근할 수 있다.)
 3. **방출** : 스레드가 자원을 방출한다.

교착 상태 특성

교착 상태는 한 시스템에 다음 네 가지 조건이 동시에 성립될 때 발생할 수 있다.

1. 상호 배제(Mutual Exclusion)

- 최소한 하나의 자원이 비공유 모드로 점유되어야 한다. 비공유 모드에서는 한 번에 한 스레드만이 그 자원을 사용할 수 있다. 다른 스레드가 그 자원을 요청하면, 요청 스레드는 자원이 방출될 때까지 반드시 지연되어야 한다.

2. 점유하며 대기(Hold-and-Wait)

- 스레드는 최소한 하나의 자원을 점유한 채, 현재 다른 스레드에 의해 점유된 자원을 추가로 얻기 위해 반드시 대기해야 한다.

3. 비선점(No Preemption)

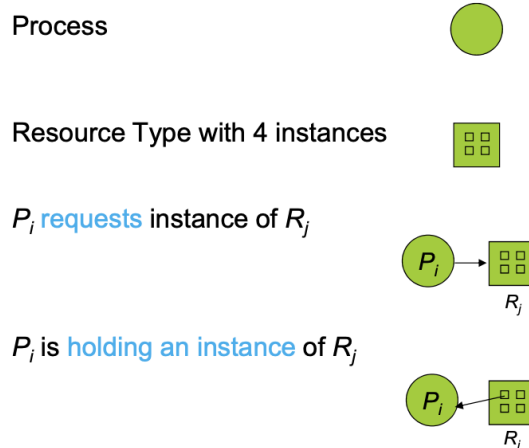
- 자원들을 선점할 수 없어야 한다. 즉, 자원이 강제로 방출될 수 없고, 점유하고 있는 스레드가 태스크를 종료한 후 그 스레드에 의해 자발적으로만 방출될 수 있다.

4. 순환 대기(Circular Wait)

- 대기하고 있는 스레드의 집합 $\{T_0, T_1, \dots, T_n\}$ 에서 T_0 는 T_1 이 점유한 자원을 대기하고, T_1 은 T_2 가 점유한 자원을 대기하고, \dots , T_{n-1} 은 T_n 이 점유한 자원을 대기하며, T_n 은 T_0 가 점유한 자원을 대기한다.

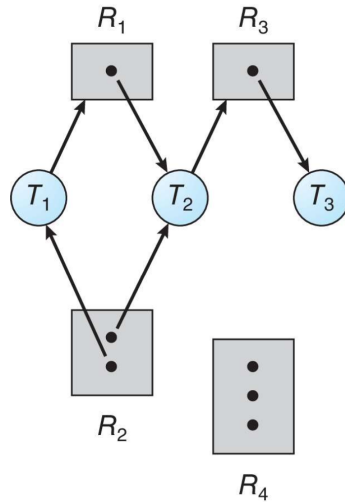
자원 할당 그래프(Resource-Allocation Graph)

- 교착 상태는 시스템 자원 할당 그래프라고 하는 방향 그래프로 더욱 정확하게 기술할 수 있다. 이 그래프는 정점(vertex) V 의 집합과 간선(edge) E 의 집합으로 구성된다. 정점 V 의 집합은 시스템 내의 모든 활성 스레드의 집합인 $T=\{T_1, T_2, \dots, T_n\}$ 과 시스템 내의 모든 자원 유형의 집합인 $R=\{R_1, R_2, \dots, R_m\}$ 의 두 가지 유형으로 구별된다.
- 방향 간선은 두 가지로 분류할 수 있다.
 - 요청 간선(request edge) : 스레드가 자원을 요청
 - 할당 간선(assignment edge) : 자원이 스레드에 할당



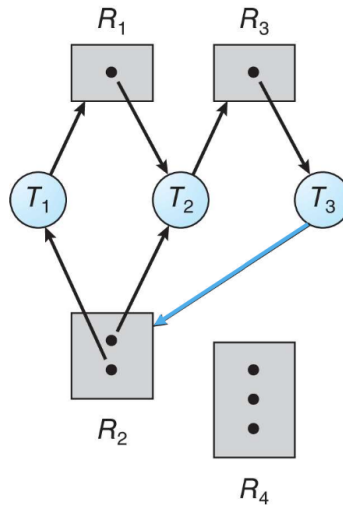
- 스레드 T_i 가 자원 유형 R_j 의 한 인스턴스를 요청하면, 요청 간선이 자원 할당 그래프 안에 삽입된다. 이 요청이 만족할 수 있으면, 요청 간선은 즉시 할당 간선으로 변환된다. 스레드가 더 자원에 대한 접근을 해야 하지 않으면 자원을 방출하고, 그 결과 할당 간선은 삭제된다.

자원 할당 그래프 [1]



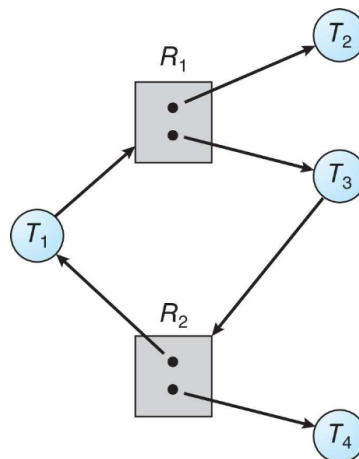
- 스레드 상태
 - 스레드 T1은 자원 유형 R2의 인스턴스 한 개를 점유하고, 자원 유형 R1의 인스턴스 한 개를 기다리며 대기한다.
 - 스레드 T2는 R1과 R2의 인스턴스를 각각 한 개씩 점유하고, 자원 유형 R3의 인스턴스 한 개를 기다린다.
 - 스레드 T3는 R3의 인스턴스 한 개를 점유하고 있다.
- 순환 구조 없음 → 데드락 발생 X
- 자원 할당 그래프의 정의로부터, 우리는 만일 그래프가 사이클(cycle)을 포함하지 않으면 시스템 내 어느 스레드도 교착 상태가 아니라는 것을 보일 수 있다. 반면, 그래프가 사이클을 포함하면 교착 상태가 존재할 수 있다.
- 만일 각 자원이 정확하게 하나의 인스턴스만을 가지면, 하나의 사이클은 교착 상태가 발생하였음을 암시한다. 만일 사이클이 각각 하나의 인스턴스만 갖는 자원 유형의 집합만을 포함한다면, 교착 상태가 발생한 것이다. 사이클에 포함된 각 스레드는 교착 상태에 있다. 이 경우에 그래프 내의 한 사이클은 교착 상태가 존재하기 위한 필요, 충분 조건이 된다.
- 만일 각 자원 유형이 여러 개의 인스턴스를 가지면, 사이클이 반드시 교착 상태가 발생했음을 의미하지는 않는다. 이 경우 그래프 내의 사이클은 교착 상태가 존재하는 데 필요조건이며 충분 조건이 되지 않는다.

자원 할당 그래프 [2]



- 두 개의 순환 사이클 → 데드락 발생

자원 할당 그래프 [3]



- 위 예에서는 교착 상태가 없다. 프로세스 T4가 자원 유형 R2의 인스턴스를 방출할 수 있음을 관찰하라. 이어 그 자원이 T3에 할당될 수 있고, 그 경우 사이클이 없어진다.



요약하자면, 자원 할당 그래프에 사이클이 없다면, 시스템은 교착 상태가 아니다. 반면에, 사이클이 있다면 시스템은 교착 상태일 수도 있고 아닐 수도 있다.

교착 상태 처리 방법

- 교착 상태가 발생하지 않도록 하기 위해 시스템은 교착 상태 예방(prevention), 혹은 회피(avoidance) 기법의 하나를 사용할 수 있다.

- 교착 상태 예방은 필요조건 중 적어도 하나가 성립하지 않도록 보장하는 일련의 방법이다. 이들 방법은 자원이 어떻게 요청될 수 있는지를 제한함으로써 교착 상태를 예방한다.
- 교착 상태 회피는 스레드가 평생 요구하고 사용할 자원에 대한 부가적인 정보를 미리 제공할 것을 요구한다, 이러한 추가적인 지식을 가지고, 운영체제는 각 요청을 위해 그 스레드가 기다려야 할지 않을지를 결정할 수 있다. 각 요구는 시스템이 현재 요청을 만족할 수 있는지, 또는 반드시 지연시켜야 하는지를 결정하기 위해, 현재 유용한 자원들과 각 스레드에 현재 할당된 자원들, 각 스레드의 미래 요청과 방출을 고려하도록 요구한다.
- 문제를 무시하고, 교착 상태가 시스템에 절대 발생하지 않는 척 한다. → Linux와 Windows를 포함해 대부분의 운영체제가 사용하는 방법
- 시스템이 교착 상태가 되도록 허용한 다음에 복구시키는 방법

교착 상태 예방(Deadlock Prevention)

교착 상태가 발생하려면 네 가지의 필요조건 각각이 성립하여야 한다.

1. 상호 배제(Mutual Exclusion)

- 상호 배제 조건이 성립되어야 한다. 즉 적어도 하나의 자원은 공유가 불가능한 자원이어야 한다. 반면에, 공유 가능한 자원들은 배타적인 접근을 요구하지 않으며, 따라서 교착 상태에 관련될 수 없다. 읽기-전용 파일이 공유 가능한 자원의 좋은 예이다. 만일 여러 스레드가 읽기-전용의 파일을 열면(open), 그들은 그 파일에 동시 접근을 허용한다.

2. 점유하며 대기(Hold and Wait)

- 시스템에서 점유하며 대기 조건이 발생하지 않도록 하려면 스레드가 자원을 요청할 때마다 다른 자원을 보유하지 않도록 보장해야 한다. 스레드가 추가의 자원을 요청할 수 있으려면, 자신에게 할당된 모든 자원을 반드시 먼저 방출해야 한다.
- 이 프로토콜은 두 가지 주요 단점이 있다.
 - 자원이 할당되었지만 장기간 사용되지 않을 수 있기 때문에 자원 이용률이 낮을 수 있다.
 - 인기 있는 여러 개의 자원이 필요한 스레드는 필요한 자원 중 적어도 하나는 항상 다른 스레드에 할당되므로 무한정 대기해야 할 수 있어 기아가 발생할 수 있다.

3. 비선점(No Preemption)

- 이미 할당된 자원이 선점되지 않아야 한다.
- 만일 어떤 자원을 점유하고 있는 스레드가 즉시 할당할 수 없는 다른 자원을 요청하면(즉, 스레드가 반드시 대기해야 하면), 현재 점유하고 있는 모든 자원이 선점된다.

즉, 이들 자원들이 묵시적으로 방출된다. 선점된 자원들은 그 스레드가 기다리고 있는 자원들의 리스트에 추가된다. 스레드는 자신이 요청하고 있는 새로운 자원은 물론 이미 점유하였던 옛 자원들을 다시 획득할 수 있을 때만 다시 시작될 것이다.

4. 순환 대기(Circular Wait)


- 모든 자원 유형에 전체적인 순서를 부여하여, 각 프로세스가 열거된 순서대로 오름차순으로 자원을 요청하도록 요구하는 것이다.

Resources must be acquired in order.
If:

first_mutex = 1
second_mutex = 5

code for **thread_two** could not be written as follows:

```
/* thread.one runs in this function */  
void *do_work.one(void *param)  
{  
    pthread_mutex_lock(&first_mutex);  
    pthread_mutex_lock(&second_mutex);  
    /**  
     * Do some work  
     */  
    pthread_mutex_unlock(&second_mutex);  
    pthread_mutex_unlock(&first_mutex);  
    pthread_exit(0);  
}  
  
/* thread.two runs in this function */  
void *do_work.two(void *param)  
{  
    pthread_mutex_lock(&second_mutex);  
    pthread_mutex_lock(&first_mutex);  
    /**  
     * Do some work  
     */  
    pthread_mutex_unlock(&first_mutex);  
    pthread_mutex_unlock(&second_mutex);  
    pthread_exit(0);  
}
```



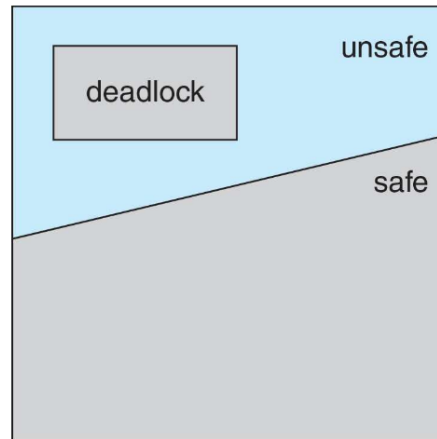
- 해당 함수에서 동시에 first_mutex와 second_mutex를 사용하기를 원하는 스레드는 반드시 first_mutex를 먼저 요청하고 다음에 second_mutex를 요청해야 한다.

→ 교착 상태 예방은 장치의 이용률이 저하되고 시스템 총처리율(throughput)이 감소한다는 단점 이 있다.

교착 상태 회피(Deadlock Avoidance)

- 자원이 어떻게 요청될지에 대한 추가 정보를 제공하는 것이다. 각 스레드의 요청과 방출에 대한 완전한 순서를 파악하고 있다면, 우리는 각 요청에 대해서 가능한 미래의 교착 상태를 피하고자 스레드가 대기해야 하는지 여부를 충족할 수 있다.
- 각 스레드가 자신이 필요로 하는 각 유형의 자원마다 최대 수를 선언하도록 요구하는 것이다. 각 스레드가 요청할 각 유형의 자원의 최대 수 정보를 미리 파악할 수 있다면, 우리는 교착 상태에 들어가지 않을 것을 보장하는 알고리즘을 만들 수 있다. 교착 상태 회피 알고리즘은 시스템에 순환 대기 상황이 발생하지 않도록 자원 할당 상태를 검사한다. 자원 할당 상태는 가용 자원의 수, 할당된 자원의 수 그리고 스레드들의 최대 요구 수에 의해 정의된다.

Safe, Unsafe, Deadlock State



- 만약 시스템이 safe 상태라면 → Deadlock 없음
- 만약 시스템이 unsafe 상태라면 → Deadlock의 가능성 있음
- Avoidance → 시스템이 절대 unsafe 상태에 들어가지 않음을 보장함

Safe State

- 시스템 상태가 안전하다는 말은 시스템이 어떤 순서로든 스레드들이 요청하는 모든 자원을(최대 요구 수를 요구하더라도) 교착 상태를 야기시키지 않고 차례로 모두 할당해 줄 수 있다는 것을 뜻한다. 즉, 시스템이 안전 순서(safe sequence)를 찾을 수 있다면 시스템은 안전하다고 말한다.

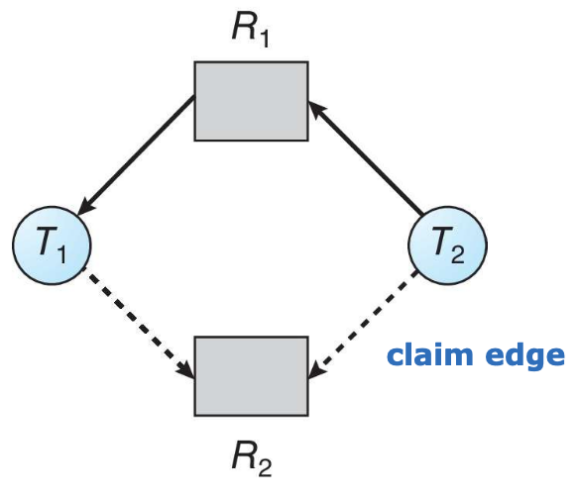
회피 알고리즘

- 자원 타입의 단일 인스턴스
 - 자원 할당 그래프의 사용
- 자원 타입의 멀티 인스턴스
 - Banker's 알고리즘 사용

자원 할당 그래프

- 자원 할당 그래프가 오직 1개의 인스턴스를 갖는 자원으로 구성되어 있다고 가정할 때, 데드락 회피를 위해 요청 가능 간선(claim edge)를 추가할 수 있다.
- $T_i \rightarrow R_j$ 은 T_i 스레드가 미래에 R_j 자원을 요청할 수 있음을 의미한다.
- claim edge를 자원 할당 그래프에 추가하여 순환 구조가 발생하는지 판단하면 시스템이 안전 상태에 있는지 확인할 수 있다. 만약 claim edge를 추가한 자원 할당 그래프에 순환 구조가 없다면 안전 상태에 있다는 뜻으로, 추후 그 요청이 들어왔을 때 허용해 줄 수 있다.

claim edge를 추가했을 때 순환 구조가 없는 경우



claim edge를 추가했을 때 순환 구조가 발견하여 unsafe 상태에 들어가는 경우

