1. 교착상태(Deadlock) 이란 무엇이고 어떤 조건들이 갖춰진 경우에 발생하는지 알려주세요.

What is the Deadlock

모든 프로세스 자신을 실행하기 위해 여러 자원(CPU, Memory, Disk, Network, ...)을 필요로 한다. 프로세스에게 자원이 부여되는 과정은 아래 순서와 같다.

1. 프로세스가 자원을 요청한다.
2. OS는 이용가능한 자원이 있다면 이를 process에게 건내주고 그렇지 않다면 process가 기다리게 끔 합니다.
3. 프로세스는 자원을 사용하고 작업이 수행된 후 할당받은 자원을 release 합니다.

Deadlock이란

- **자원을 기다리는 waiting 프로세스들이 자원을 점유하며 작업을 수행하고있는 executing 프로세스들의 작업이 종료되고 할당된 자원을 release 하기를 기다리는 상황에서**
- **executing 프로세스들도 다른 프로세스가 자원을 release 하기를 기다리고 있는 상황으로 인해 모든 프로세스들이 자원 요구 때문에 작업 수행이 무한히 미뤄지는 상황을 의미합니다.**

Necessary conditions for Deadlock

Deadlock은 시스템에서 다음의 4가지 조건이 모두 동시에 발생하는 경우에만 일어날 수 있다.

1. **Mutual Exclusion**

   저번 멘토링 주제인 Race Condition에서 프로세스/스레드 간에 동기화를 유지하는 방법을 통해 다루었던 Mutual Exclusion(상호 배제)이다. 하나의 자원을 둘 이상의 프로세스/스레드가 동시에 사용할 수 없다는 의미이다.

   하나의 자원을 둘 이상의 프로세스/스레드가 동시에 사용할 수 있다면 자원을 기다리는 상황은 일어나지 않겠죠?

2. **Hold and Wait**

   현재 하나 이상의 자원을 점유하고 있는 프로세스가 추가적으로 다른 프로세스에 의해 점유되어져있는 자원을 사용하도록 요청하고 기다리는 상황.

   이미 실행중인 프로세스가 점유되어지지 않는 자원을 추가로 가져간다면 작업이 정상적으로 끝나겠지만 다른 프로세스가 점유하고 있는 자원을 요구하기 때문에 기다리는 상황이 발생할 수 밖에 없겠죠?

   이게 연쇄적으로 일어난다면 더 문제구요. (occur Circular Wait)

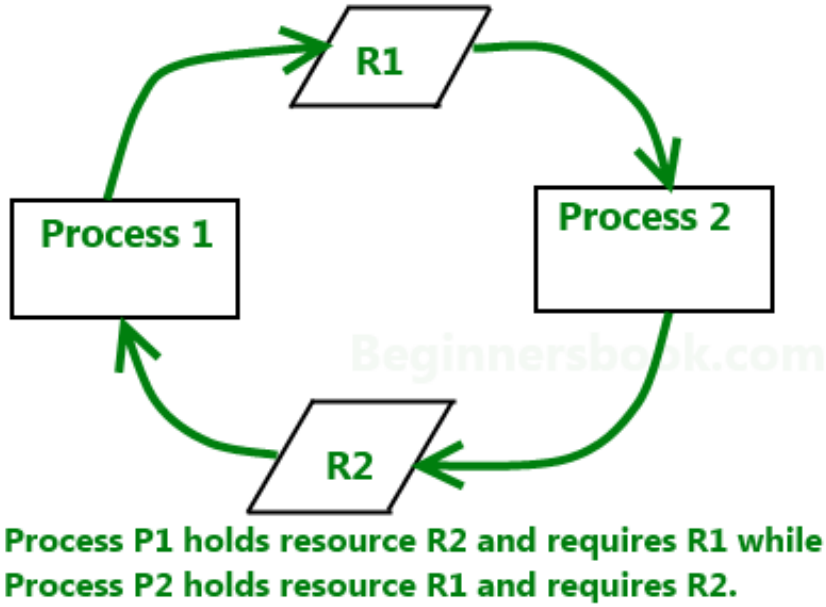
3. **No Preemption**

   프로세스가 다른 프로세스에 점유되어진 자원을 강제로 가져올 수 없다는 즉, 선점할 수 없다는 의미입니다.

   다른 프로세스의 자원을 강제로 가져올 수 없으니 해당 프로세스의 실행이 완료된 후 자원을 release 할 때 까지 기다릴 수 밖에 없겠죠?

4. **Circular wait**

   모든 프로세스는 원형 형태로 다른 프로세스가 점유하고 있는 자원이 release 되기를 기다리는 형태를 의미한다.



쉽게 설명하면 어떠한 둘 이상의 프로세스도 하나의 자원을 기다리지 않고 하나의 프로세스가 다른 하나의 자원을 기다리는 형태를 그림으로 표현해보니 꼭 원형과 같다라는 의미입니다.

즉, 시스템에서 Process/Thread가 Resource를 사용하는 과정에서 Mutual Execlusion, Hold and Wait, No Preemption, Circular Wait 이 모두 동시에 발생하는 경우에 Deadlock이 발생합니다.

---

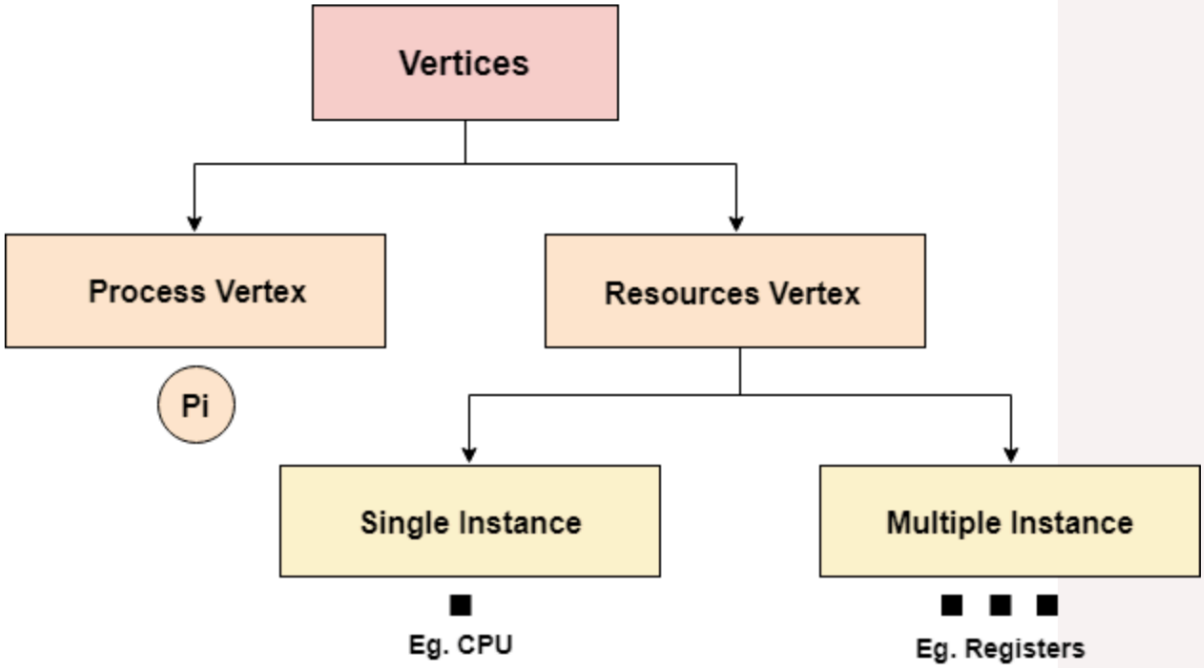
2. 자원 할당 그래프(Resource Allocation Graph)를 사용해서 Deadlock을 표현해보시오.

Resource Allocation Graph

자원 할당 그래프(Resource allocation graph)는 시스템 자원의 상태를 그림으로 표현한 그래프를 의미한다. 이 그래프에선 자원을 점유하거나 기다리고 있는 프로세스 모두를 완전히 표현해준다.

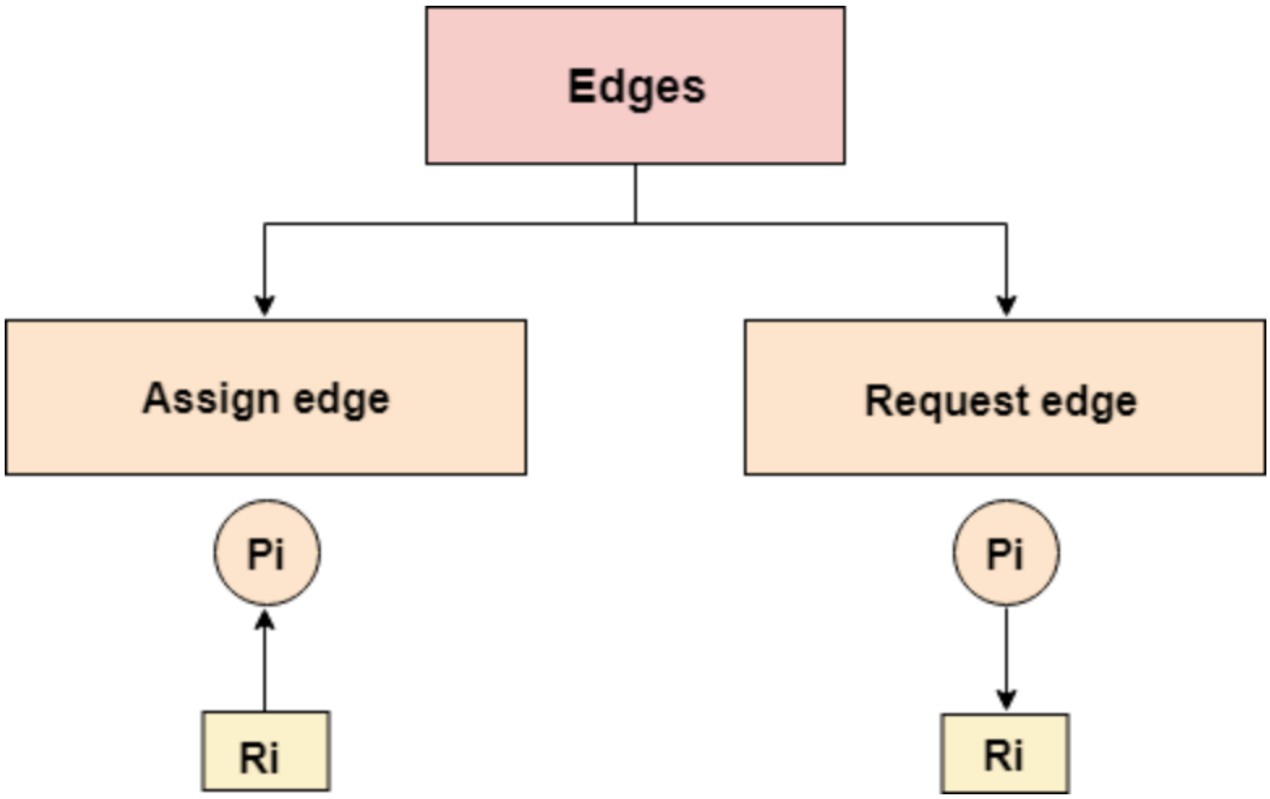
또한 이용가능하거나 프로세스에 의해 사용되어지는 모든 자원의 인스턴스에 대한 정보도 표현한다.

Resource allocation graph에서 프로세스는 원으로 표현되어지는 반면에 resource는 정사각형으로 표현되어진다. 아래 그림을 참고해 보자.



정점들은 주로 자원과 프로세스 두 가지 타입으로 나뉘고 이들 각각은 다른 형태의 모양으로 표현되어진다. 원은 process를 나타내는 반면에 정사각형은 resource를 나타낸다.

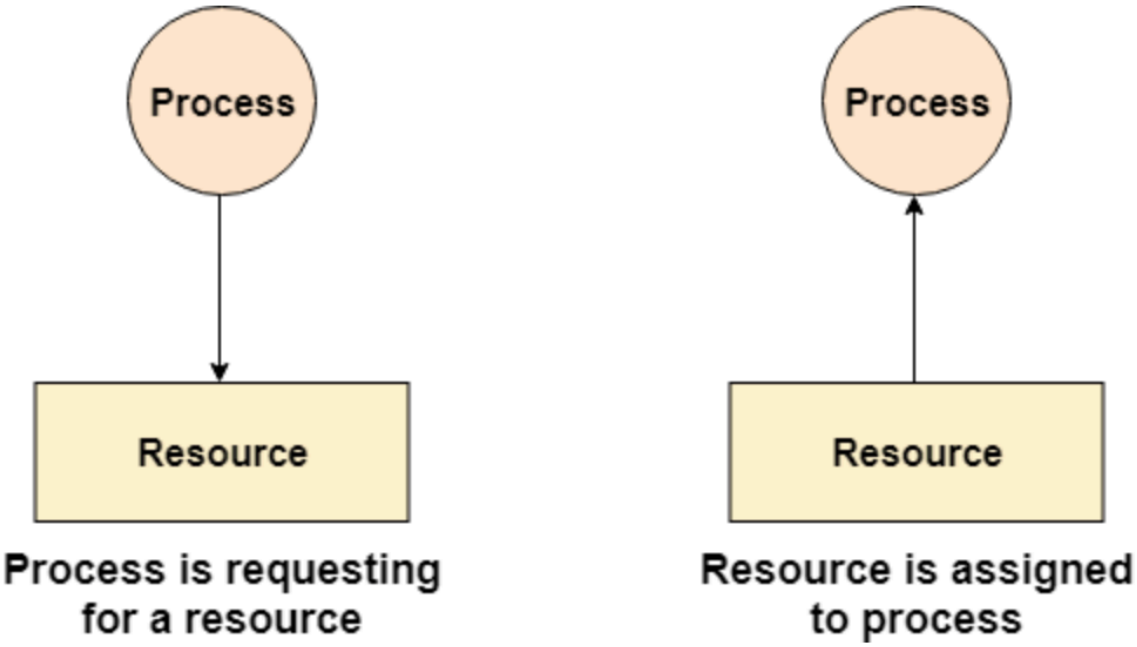
resource는 하나 이상의 인스턴스를 가질 수 있으며, 각 인스턴스는 정사각형안에 점으로써 표현되어질 것이다.



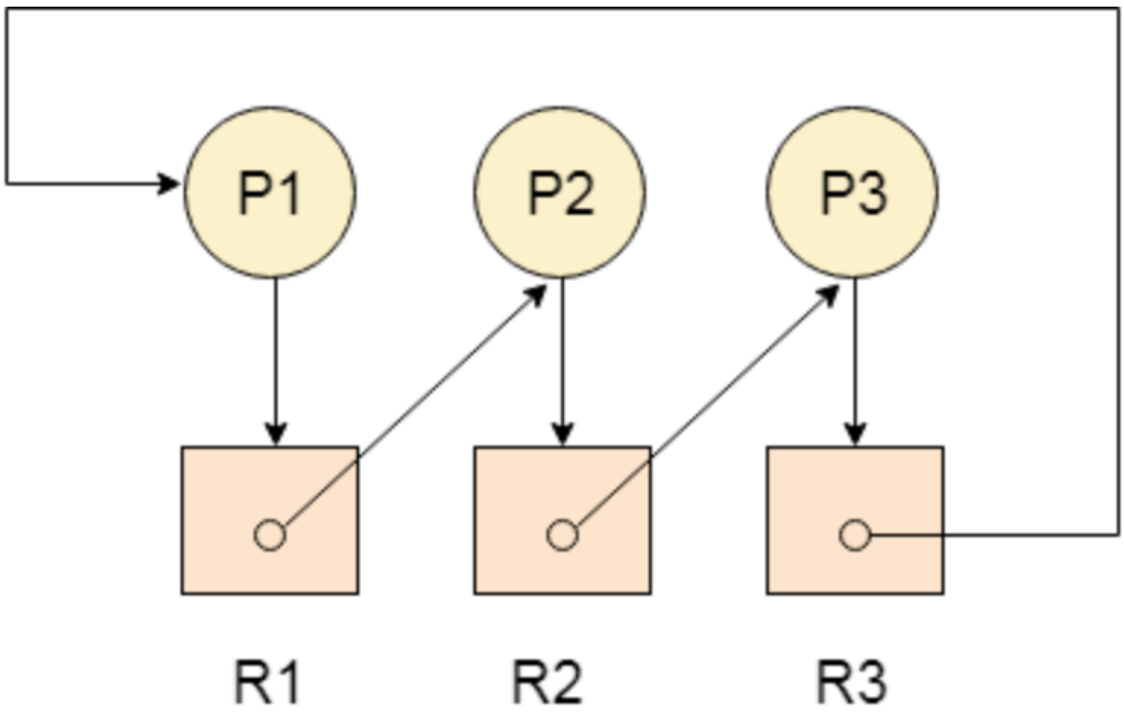
Resource allocation graph에서 Edges는 두가지 타입으로 존재할 수 있다. 하나는 할당을 의미하며, 다른 하나는 자원을 위해 process가 기다리고 있는 상태를 표현한다. 위의 이미지는 이러한 상태들을 보여준다.

만약 화살표의 꼬리가 리소스의 인스턴스에 붙어 있고 머리가 프로세스에 붙어있다면 자원이 할당된 프로세스를 의미합니다.

화살표의 꼬리가 프로세스에 붙어 있고 머리가 자원에 붙어 있따며 이는 자원을 기다리는 프로세스를 의미합니다.



Deadlock Represntation using Resource Allocation Graph



이 그림이 Deadlock인 이유

1. Mutual Exclusion : 하나의 자원에 동시에 둘 이상의 프로세스가 접근할 수 없다.
2. Hold and Wait : 현재 하나 이상의 자원을 점유하고 있는 프로세스(p1, p2, p3)가 추가적으로 다른 프로세스에 의해 점유되어져있는 자원(r1, r2, r3)을 사용하도록 요청하고 기다리는 상황.

3. No Preemption : 선점이 안되는 건 이런 Resource Allocation Graph 와 같은 System Status Snapshot으로는 알 수 없으니 가정하자.
4. Circular wait : 모든 프로세스(p1, p2, p3)는 원형 형태로 다른 프로세스가 점유하고 있는 자원(r1, r2, r3)이 release 되기를 기다리는 형태

---

3. Deadlock을 다루기 위한 방법(ignorance, prevention, ...)은 이와 같이 여러가지가 있는데 각각의 장단점은 무엇이고 현재 어떤 방법이 가장 많이 쓰이고 있고 왜 그런지 알려주세요.

3-1. Deadlock Ignorance

Deadlock Ignorance는 Deadlock을 해결하는 여러 방법 중 가장 널리 사용되어지는 방법이다. 이는 많은 os 시스템에서 주로 사용되어집니다. 이러한 접근 방법에서 OS는 deadlock이 절대 일어나지 않는다고 가정하고 발생 했을 경우 이를 무시합니다. 이러한 접근 방법은 단일 사용자가 브라우징이나 간단한 작업들을 처리할 때 사용하는 시스템에서 적절합니다.

정확성과 성능 사이에는 항상 tradeoff가 존재합니다. Window와 Linux 같은 OS는 주로 성능에 집중합니다. 그러나 deadlock handling mechanism을 매 시간 돌리는 방법을 사용하거나, 교착 상태가 100번중 한번 발생 한다면 시스템의 성능은 떨어지게 됩니다. 그래서 매번 deadlock handling mechnism을 사용하는 것은 불필요합니다.

Deadlock Handling 방법으로 Deadlock Ignorance를 채택하는 시스템에서 사용자는 Deadlock이 발생하는 경우 간단히 재시작하는 방법으로 이 상황을 해결할 수 있습니다. 역시 이 방법들은 Window와 Linux가 주로 사용하빈다.

시스템이 Deadlock을 다루는 방법으로 Deadlock Ignorance를 사용함으로써 시스템의 성능을 향상시킬 수 있겠지만 시스템이 Deadlock으로 인해 갑자기 종료되는 경우가 생길 수 있음으로 Data Correctness는 떨어진다.

3-2. Deadlock Prevention

Deadlock은 Mutual Exclusion, hold and wait, No preemption, circular wait이 동시에 발생할 때 일어난다. 만약 이 4가지 중 하나라도 발생하지 않을 경우 deadlock은 시스템에서 절대 발생하지 않는다.

즉, 이 접근법의 아이디어는 Deadlock 발생에 필요한 4가지 조건 중 최소한 하나의 조건이라도 충족되지 못하게 만들어 Deadlock을 예방하는 방법을 의미한다.

1. Mutual Exclusion
    - Mutual Exclusion을 violate하면서 병렬/동시성 작업을 진행한다면 당연히 Race Condition이 발생해 작업의 일관성 및 정확성이 깨진다.
2. Hold and Wait
    - Process가 자원을 점유하면서 작업을 수행하는 동안 다른 자원을 기다리지 않게 처음부터 작업 수행간에 필요한 모든 자원을 가지고 초기화 한 후 작업을 실행하도록 하는 방법.

      현실적으로 매우 어렵다. 만약 이런 방법을 사용하게 된다면 우리는 Virtual Memory 와 같은 가상화 기법을 사용할 수 없게되 매우 제한적인 컴퓨팅이 될 것이다.
3. No Preemption
    - 자원을 preemptive 하게 다른 프로세스로 부터 가져온다면 Deadlock을 예방할 수 있다. 하지만 이 또한 적절한 방법이 아니다... 다른 프로세스에서 현재 사용중인 리소스를 가져와 사용한다면 자원을 뺏긴 프로세스는 작업의 일관성을 잃기 때문에 좋은 접근방식이 아니다.
4. Circular Wait (가장 실질적으로 구현할 수 있는 방법)

- 각 자원에 우선순위를 할당해 프로세스가 우선순위가 낮은 자원을 가져올 수 없게 만들어 모든 프로세스가 자원을 요청해서 원형 대기가 발생하지 않도록 한다.

  다른 프로세스에서 사용되어지고 있는 자원을 요청하지 않게 한다든지, cycle이 형성되지 않게 전체적인 process resource usage snapshot을 뽑는다던지 여러 방법이 있지만 역시 성능이 떨어진다.

3-3. Deadlock Avoidance

Deadlock Avoidance를 위해서 OS는 프로세스에 자원을 할당할 때마다 시스템이 안전한 상태에 있는지 아닌지를 매번 점검한다. `(Banker Algorithm)`

시스템이 safe state일 때 process는 정상적으로 동작할 것이며, unsafe state로 전이가 되어진다면 OS는 이전 스텝들을 되추적할 것이다.

OS가 프로세스에 자원을 할당할 때마다 System이 Safe한지 Unsafe한지 매번 검증해야하고 Unsafe하다면 추가적인 조치를 취해야하기 때문에 Deadlock Avoidance 역시 정확성은 높아지지만 성능이 좋지 않아진다.

3-4. Deadlock detection and recovery

System에서 Deadlock이 언제든지 일어날 수 있음을 인정하고 지속적으로 이를 detection하고 Deadlock이 발견될 시 이를 Recovery 하는 접근방법을 의미한다.

이를 위해 시스템은 Deadlock을 주기적으로 Dection 해야하고, Deadlock이 발생시 Recovery를 위한 해결책을 base로 구현하면서 시스템을 운영해야 하기 때문에 역시... Correctness는 올라갈지는 몰라도 Performance는 떨어진다.

---

* The Good Contents for Deadlock Studying

https://www.javatpoint.com/os-deadlocks-introduction 이 웹사이트의 빨간 네모박스의 주제만 다 읽는다면 이번 과제는 쉽게 할 수 있을 겁니다.