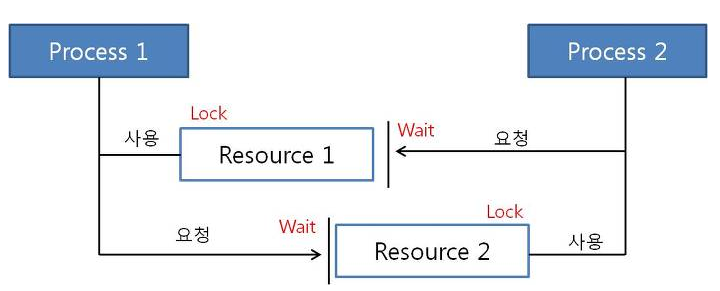
운영체제 OS

**데드락 (DeadLock) // 교착 상태**

: 두 개 이상의 작업이 서로 상대방의 작업이 끝나기 만을 기다리고 있기 때문에 결과적으로 아무것도 완료되지 못하는 상태를 가리킨다. 다중 스레드 프로그래밍 환경에서 발생할 수 있는 문제.



**멀티스레딩에 대한 이해**

- 멀티스레딩 환경에서 변수 a=0 일 때, a=a+1 작업을 스레드 두 개로 각각 천만 번 수행한다고 가정했을 때 그 값은 a는 2천만이 될 것으로 기대되지만 실제는 그렇지 않다.

1. a값을 불러온다.
2. a값에 1을 더한다.
3. 더한 값을 다시 a에 저장한다.

이 과정을 순차적으로 진행한다고 기대하지만 순차적이지 않고 두 스레드 모두 a=0을 가져올 수 있다. 따라서 두 개의 스레드가 교차적으로 한 번 수행되었을 때 a는 2가 아닌 1이 저장되게 된다.

**교착 상태(Deadlock)의 조건(4가지 필요조건을 충족시켜야 함)**

1. 상호배제(Mutual exclusion) : 프로세스들이 필요로 하는 자원에 대해 배타적인 통제권을 요구한다.  
   즉, 자원 자체를 동시에 쓸 수 없는 경우를 일컫는다.
2. 점유대기(Hold and wait) : 프로세스가 할당된 자원을 가진 상태에서 다른 자원을 기다린다.
3. 비선점(No preemption) : 프로세스가 어떤 자원의 사용을 끝낼 때까지 그 자원을 뺏을 수 없다.
4. 순환대기(Circular wait) : 각 프로세스는 순환적으로 다음 프로세스가 요구하는 자원을 가지고 있다.

이 조건 중에서 한 가지라도 만족하지 않으면 교착 상태는 발생하지 않는다. 이중 순환대기 조건은 점유대기 조건과 비선점 조건을 만족해야 성립하는 조건이므로, 위 4가지 조건은 서로 완전히 독립적인 것은 아니다.

⇒ 즉, 네 가지 조건 중 하나라도 성립하지 않도록 만든다면 교착 상태를 해결할 수 있다.

**데드락의 처리**

1. 교착 상태 예방 : 교착 상태가 되지 않도록 보장하기 위하여 교착 상태를 예방,방지
2. 회피 : 검사 후 데드락에 빠질 가능성이 없을 경우에만 자원을 할당하며 회피하는 방법
3. 교착 상태 탐지, 회복 : 교착 상태가 되도록 허용한 다음에 회복시키는 방법
4. 교착 상태 무시 : 대부분의 시스템은 교착 상태가 잘 발생하지 않으며, 교착 상태 예방, 회피, 탐지, 복구하는 것은 비용이 많이 든다.

**데드락 예방(Prevention)**

: 교착 상태 발생 조건 중 하나를 제거함으로써 해결하는 방법, 자원의 낭비가 심하다

1. 상호배제(Mutual exclusion) : 여러 개의 프로세스가 공유 자원을 사용할 수 있도록 한다.  
   =>하지만 많은 경우가 한 번에 한 프로세스만 사용하게 하므로 1번 조건 제거는 어렵다.
2. 점유대기(Hold and wait) : 프로세스가 실행되기 전 필요한 모든 자원을 할당한다.
3. 비선점(No preemption) : 자원을 점유하고 있는 프로세스가 다른 자원을 요구할 때 점유하고 있는 자원을 반납하고, 요구한 자원을 사용하기 위해 기다리게 한다.
4. 순환대기(Circular wait) : 자원에 고유한 번호를 할당하고, 번호 순서대로 자원을 요구하도록 한다.  
   => 대부분의 교착상태 방지 알고리즘은 4번 조건, 즉 대기 상태의 사이클이 발생하는 일을 막는 데 초점이 맞춰져 있다.

**데드락 회피(Avoidance)**

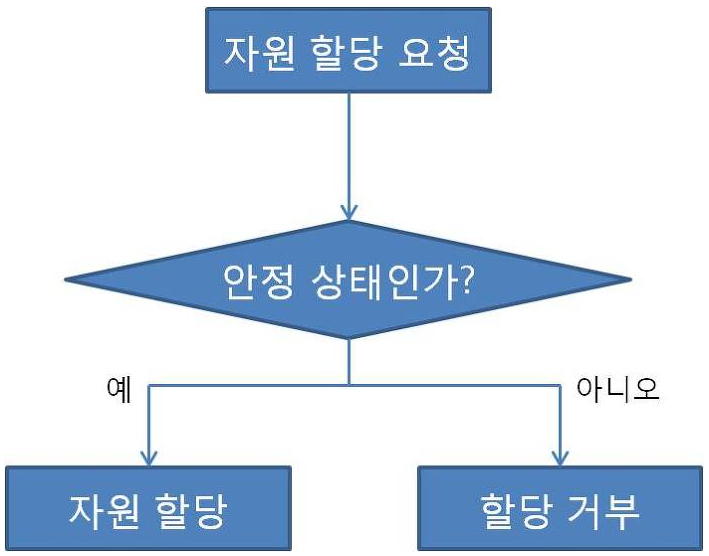
: 교착 상태가 발생하면 피해나가는 방법이다.

* 은행원 알고리즘(banker’s algorithm)

: E,J,Dijkstra가 제안한 방법으로, 은행에서 모든 고객의 요구가 충족되도록 현금을 할당하는 데서 유래한 기법이다.

프로세스가 자원을 요구할 때 시스템은 자원을 할당한 후에도 안정 상태로 남아있게 되는지를 사전에 검사하여 교착 상태를 회피하는 기법

안정 상태에 있으면 자원을 할당하고, 그렇지 않으면 다른 프로세스들이 자원을 해지할 때까지 대기함



**데드락 탐지 (Detection)**

: 자원 할당 그래프를 통해 교착 상태를 탐지할 수 있다.

자원을 요청할때마다 자원 할당 알고리즘을 실행하면 오버헤드가 발생..

리소스가 single instance일 경우 => wait-for그래프를 주기적으로 확인하면서 어떤 프로세스가 리소스를 wait하고 있는가 체크한다. (사이클이 발생했다면 데드락이 발생한 것)

Multiple instance일 경우엔 detection algorithm을 사용(banker’s algorithm과 비슷)

**데드락 회복(Recovery)**

: 교착 상태를 일으킨 프로세스를 종료하거나, 할당된 자원을 해제함으로써 회복, rollback (이 때 무한 rollback으로 starvation이 발생할 가능성이 높다)

프로세스를 종료하는 방법

1. 교착 상태의 프로세스를 모두 중지
2. 교착 상태가 제거될 때까지 한 프로세스 씩 중지

자원을 선점하는 방법

1. 교착 상태의 프로세스가 점유하고 있는 자원을 선점하여 다른 프로세스에게 할당하며, 해당 프로세스를 일시 정지 시키는 방법
2. 우선 순위가 낮은 프로세스, 수행된 횟수가 적은 프로세스 등을 위주로 프로세스의 자원을 선점

결론 : 그래프 알고리즘을 이용하여 교착 상태를 검출해낼 수 있으나 n개의 락을 이용할 경우 O(n3)의 알고리즘이 되어 상당한 성능 저하를 일으킨다.

그러한 연유로 보통의 범용 운영체제는 데드락 방지를 위한 어떠한 처리를 크게 고려하지 않는 편. 가능한 한 교착이 잘 일어나지 않게끔 설계하겠지만 능동적으로 교착 방지 알고리즘을 적용할 경우 일어나는 성능 저하가 가끔 일어나는 교착 상태보다 더 심각한 문제라고 판단하기 때문이다. 그래서 심각한 교착 상태가 일어나는 경우 그냥 사용자가 알아서 리부팅해야 한다. 이와는 반대로 이런 알고리즘을 써서라도 반드시 교착 상황을 방지해야 하는 운영체제도 있는데 바로 **차량/항공기 전장 운영체제**이다. 교착이 일어나서 시스템이 뻗는 경우 대형참사로 이어지기 때문.