Ch 8.6

8.6 Deadlock Avoidance

- deadlock prevention은 low device utilization과 reduced system throughput이라는 부작용이 존재한다.
- deadlock을 피하기 위한 또 하나의 방법은 **사전에 미리 resource request들에 대한 정보를 파악해서 deadlock을 피할 수 있는 시나리오를 짜는 방법**이다.
- 알고리즘에 따라 필요한 정보의 종류와 양이 다르지만, 가장 단순하면서도 유용한 방법은 **각 thread가 실행되는데 필요할 것으로 예상되는 최대 resource 개수를 미리 파악**해두는 것이다. 이 정보를 파악하면 운영체제는 resource-allocation state을 수시로 점검해서 deadlock이 절대로 일어나지 않도록 한다.
    - **resource-allocation state**: 각 thread가 현재 점유하고 있는 resource 개수, 현재 점유되지 않고 남아있는 resource 개수, 그리고 각 thread가 최대로 필요로한 resource 개수에 대한 정보

8.6.1 Safe State

- **현재 상황에서 교착 상태가 일어나지 않는 safe sequence가 존재하면 이를 safe state이라고 한다.** safe sequence가 존재하지 않으면 unsafe state이라고 하며, **unsafe state은 deadlock으로 이어질 가능성이 있는** 상태이다. unsafe state이 deadlock으로 이어지는지의 여부는 thread들의 행동에 따라 결정되며, thread들의 행동에 따라 safe state에서 unsafe state으로 변경될 수도 있다.
- 따라서 deadlock avoidance algorithm들은 thread들이 resource request를 할 때마다 safe state으로 남을 수 있는 방향으로 결정(request를 바로 승낙 또는 wait)을 내릴 수 있도록 판단하는 일을 한다.

8.6.2 Resource-Allocation-Graph Algorithm

- 이 알고리즘은 **resource 종류별로 개수가 하나 밖에 없을 때** 적용할 수 있다.
- resource allocation graph에 thread가 실행되기 전에 미래에 일어날 수 있는 모든 잠재적인 resourse request들을 claim edge로 표기해서 **잠재적인 deadlock 상황을 사전에 막는 방법**이다.

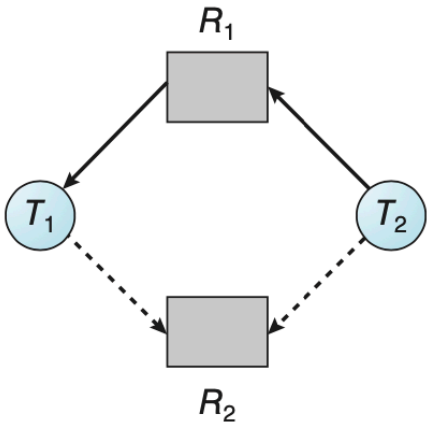

Figure 8.9 Resource-allocation graph for deadlock avoidance.

8.6.3 Banker's Algorithm

- banker's algorithm은 **resource 종류별 개수가 여러개일 때도 적용**할 수 있는 일반적인 방법이다.
- 하나의 thread가 resource request를 보내면, 운영체제는 이 요청을 받았을 때의 state를 가정해서 safety algorithm을 돌렸을 때 안전하다고 판단이 되면 할당하고 안전하지 않다고 판단이 되면 기다리게 만드는 방법이다.
- 필요한 자료 구조
  - **n**: thread의 개수, **m**: 자원 종류의 개수
  - **Available**: 모든 자원 m에 대한 사용 가능한 개수를 나타낸 m 길이의 벡터
  - **Max**: 각각의 thread n개가 제시한 모든 자원 m개에 대한 최대 요구 개수를 나타낸 n * m 행렬
  - **Allocation**: 각각의 thread n개에게 현재 할당된 모든 자원 m에 대한 개수를 나타낸 n * m 행렬
  - **Need**: 각각의 thread n개가 추가적으로 필요로하는 모든 자원 m에 대한 개수를 나타낸 n * m 행렬

8.6.3.1 Safety Algorithm

- safety algorithm을 사용해서 현재 state가 safe state인지 아닌지 판단한다.
  - 알고리즘을 돌려서 safe sequence가 존재하는지 찾는다. 존재한다면 safe state인 것으로 판단할 수 있다.

8.6.3.2 Resource-Request Algorithm

- 새로운 resourse request가 들어왔을 때, 현 resource allocation state에서 이 request를 들어 줄 수 있는지 판단한다.
  - 요청한 request가 Need보다 작거나 같은지
  - 요청한 request가 Available보다 작거나 같은지
  - 위 두 조건을 만족하면 request을 들어주고 그에 따른 Available, Allocation, Need를 갱신한다.