13-2. 교착 상태 해결 방법

교착 상태 해결 방법

예방　　회피　　검출 후 회복

교착 상태 예방

교착 상태가 일어나지 않도록

교착 상태 발생 조건에 부합하지 않게 자원을 분배하는 방법

교착 상태 예방 : 상호 배제 제거

모든 자원을 공유 가능하게 만든다.

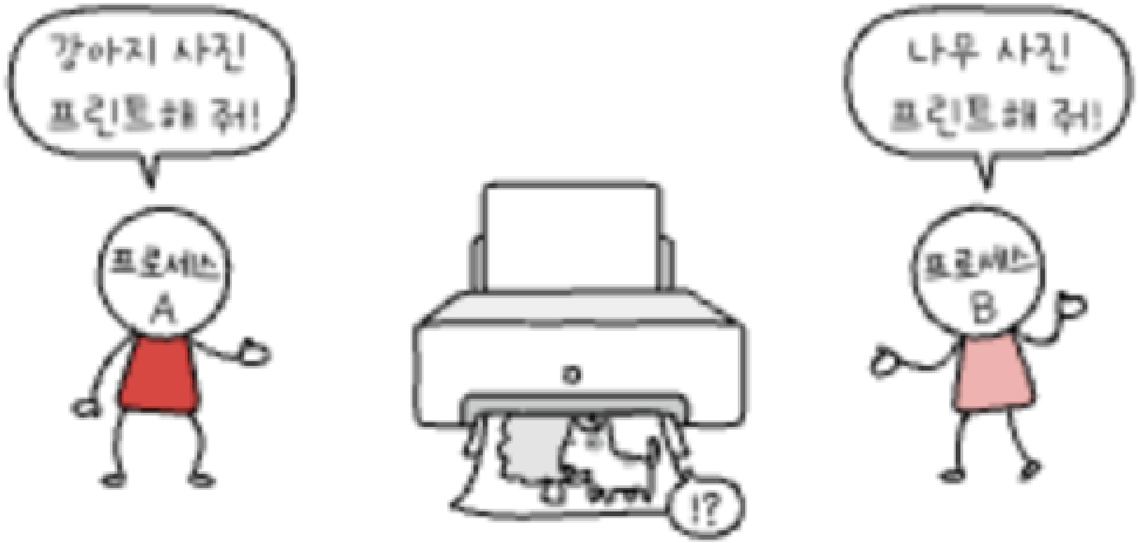
이론적으론 가능, 현실적으로 모든 자원의 상호 배제를 없애기는 어렵다.

교착 상태 예방 : 점유와 대기 제거

무조건 2개 들거나 아무것도 안들거나

자원의 활용도가 낮아지고, 자원을 많이 사용하는 프로세스는 동시에 자원을 사용할 타이밍 확보가 어렵다.

교착 상태 예방 : 비선점 제거

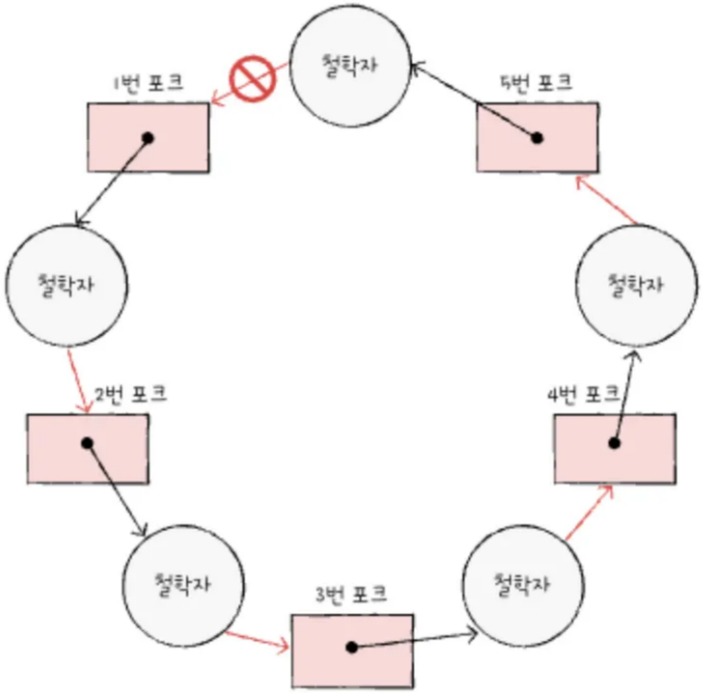


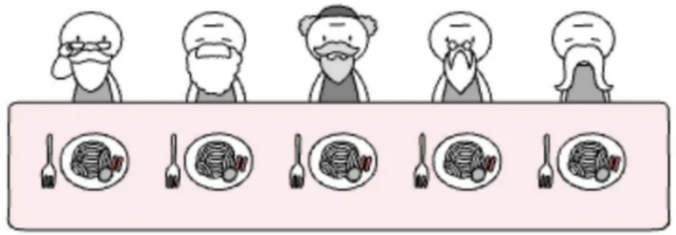
프린트 중 다른 프로세스가 프린터 자원을 빼앗아 사용한다면?

다른 그림이 한 페이지에 나오게 된다.

그래서 비선점 제거는 범용성이 떨어지는 방안이다.

교착 상태 예방 : 원형 대기 조건 제거





컴퓨터 시스템 내의 수많은 자원에 번호를 붙이는 일은 간단하지 않다.

자원에 어떤 번호를 붙이는지에 따라 특정 자원의 활용률이 떨어질 수 있다.

교착 상태 회피

교착 상태가 발생하지 않을 정도로만 조심히 자원을 할당하는 방식.

교착 상태를 한정된 자원의 무분별한 할당으로 인해 발생하는 문제로 간주.

교착 상태 회피

안전 상태: 교착 상태가 발생하지 않고 모든 프로세스가 정상적으로 자원을 할당받고 종료되는 상태

불안전 상태: 교착 상태가 발생할 수도 있는 상황

안전 순서열: 교착 상태 없이 안전하게 프로세스들에 자원을 할당할 수 있는 순서

교착 상태 회피

프로세스	요구량	현재 사용량
P1	10	5
P2	4	2
P3	9	2

- 할당 가능 자원: 12
- 할당한 자원: 9
- 남은 자원: 3

P2 -> P1 -> P3 라는 안전 순서열 존재

교착 상태 회피

프로세스 P1, P2, P3가 모두 최대로 자원을 요구한 최악의 상황

프로세스	최대 요구량	현재 사용량
P1	10	5
P2	4	2 + 2
P3	9	2

- 할당 가능 자원: 12
- 할당한 자원: 9 + 2 = 11
- 남은 자원: 3 - 2 = 1

프로세스	최대 요구량	현재 사용량
P1	10	5
~~P2~~	~~4~~	~~2 + 2~~
P3	9	2

종료, 자원 반환

- 할당 가능 자원: 12
- 할당한 자원: 11 - 4 = 7
- 남은 자원: 1 + 4 = 5

교착 상태 회피

프로세스 P1, P2, P3가 모두 최대로 자원을 요구한 최악의 상황

프로세스	최대 요구량	현재 사용량
P1	10	5 + 5
P2	4	2 + 2
P3	9	2

- 할당 가능 자원: 12
- 할당한 자원: 7 + 5 = 12
- 남은 자원: 5 - 5 = 0

프로세스	최대 요구량	현재 사용량
P1	10	5 + 5
P2	4	2 + 2
P3	9	2

종료, 자원 반환

- 할당 가능 자원: 12
- 할당한 자원: 12 - 10 = 2
- 남은 자원: 0 + 10 = 10

교착 상태 회피

운영체제가 P3에 먼저 자원을 하나 내준 상황

프로세스	최대 요구량	현재 사용량
P1	10	5
P2	4	2
P3	9	3

- 할당 가능 자원: 12
- 할당한 자원: 10
- 남은 자원: 2

프로세스	최대 요구량	현재 사용량
P1	10	5
~~P2~~	~~4~~	~~2 + 2~~
P3	9	3

종료, 자원 반환

- 할당 가능 자원: 12
- 할당한 자원: 10 + 2 = 12 → 12 - 4 = 8
- 남은 자원: 2 - 2 = 0 → 0 + 4 = 4

교착 상태 발생 위험

교착 상태 검출 후 회복

프로세스들이 자원을 요구할 때마다 모두 할당하며,
교착 상태 발생 여부를 주기적으로 검사한다

교착 상태 검출 후 회복 : 선점을 통한 회복

교착 상태 해결까지 한 프로세스씩 자원을 몰아주는 방식.

다른 프로세스로부터 자원을 강제로 빼앗고 한 프로세스에 할당.

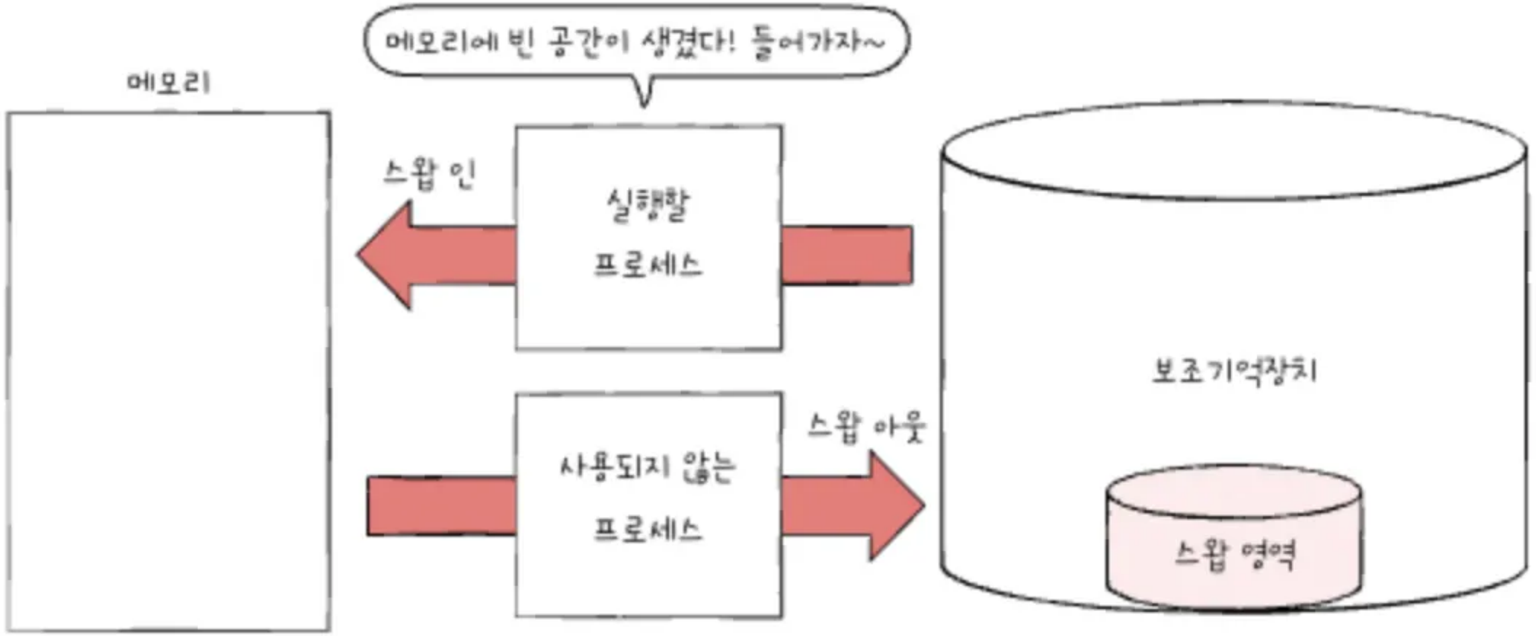
교착 상태 검출 후 회복 : 프로세스 강제 종료를 통한 회복

교착 상태에 놓인 프로세스를 모두 강제 종료할 수도,

교착 상태가 없어질 때까지 한 프로세스씩 강제 종료할 수도 있다.

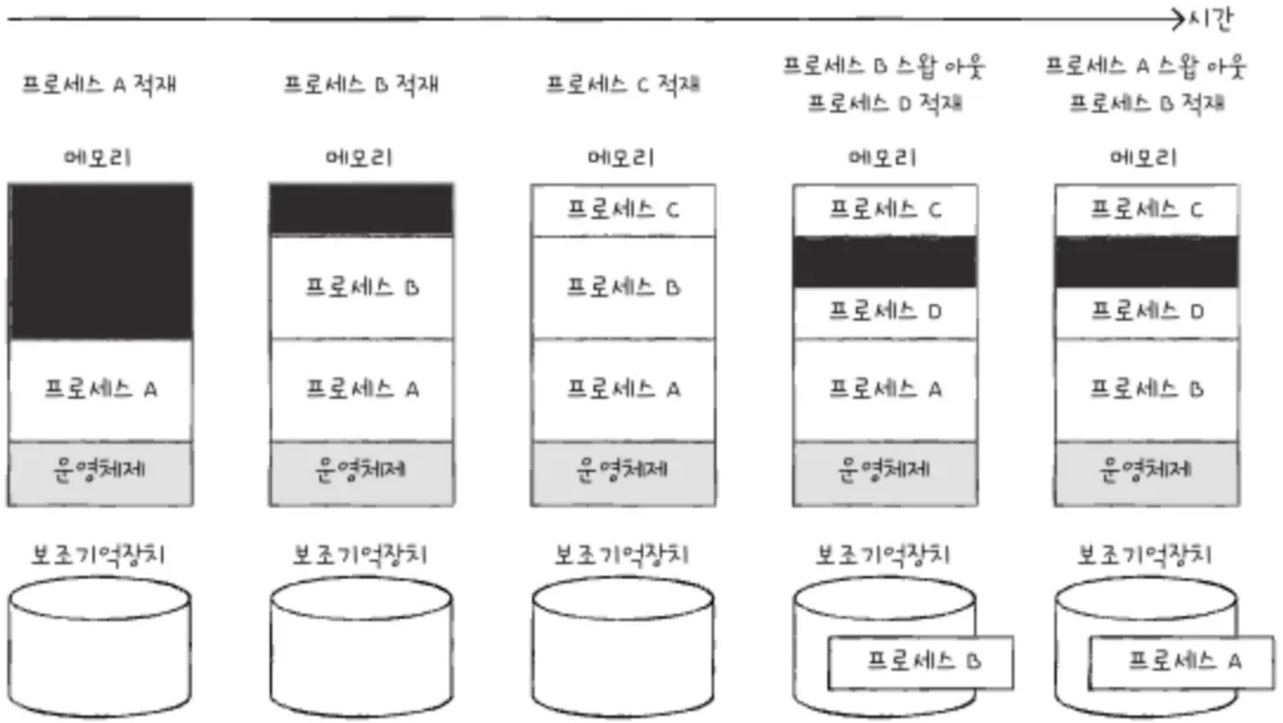
14-1. 연속 메모리 할당

스와핑



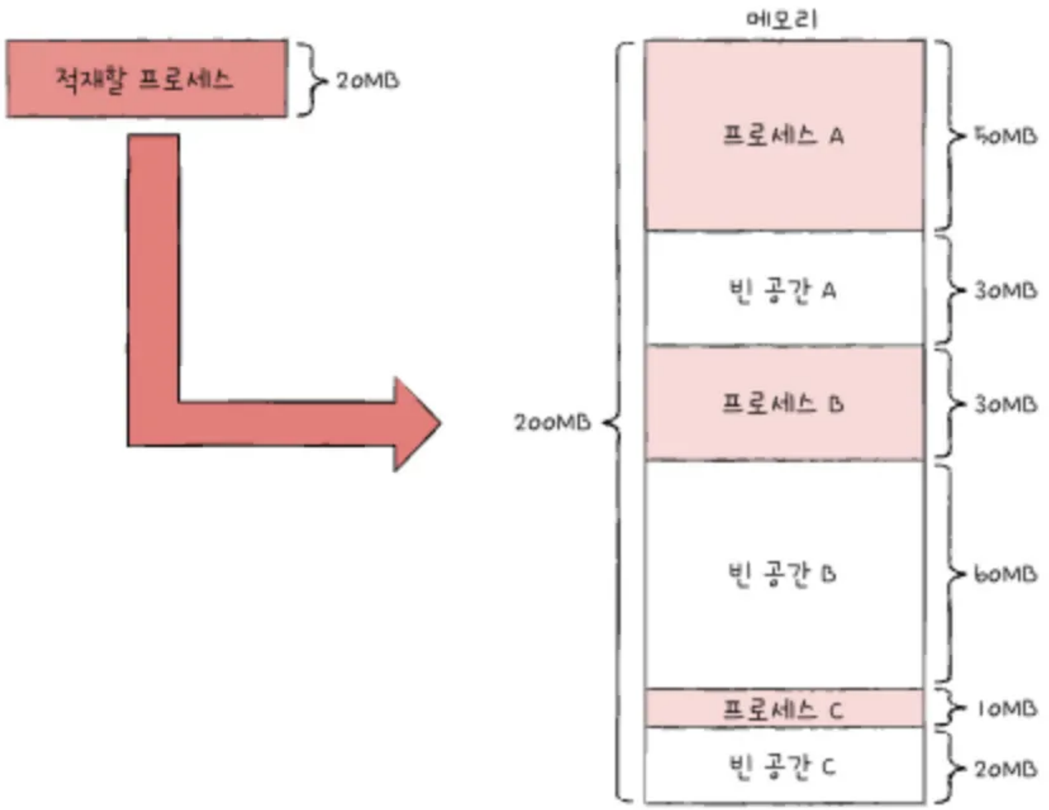
메모리에 현재 실행되지 않는 프로세스들을 보조기억장치로,
실행될 프로세스들을 메모리로 옮기는 방법

스와핑



스와핑을 통해 네 개의 프로세스를 모두 실행할 수 있음.

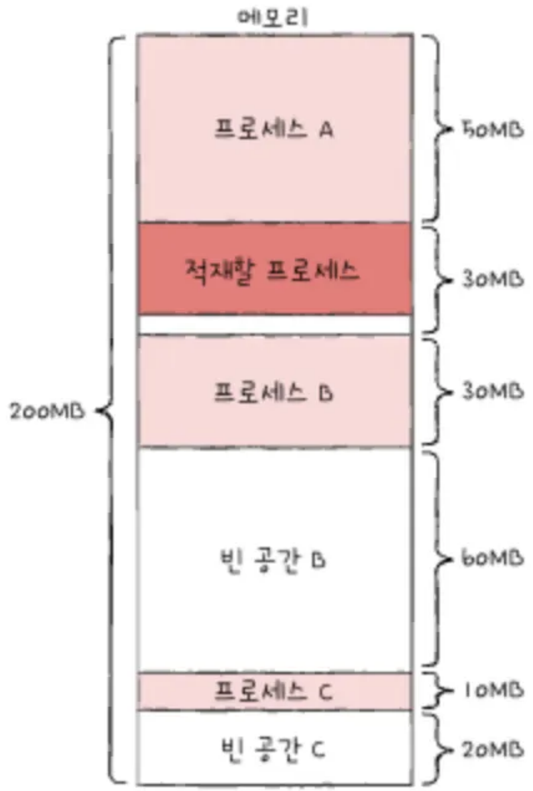
메모리 할당



프로세스는 메모리 내 빈 공간 중

최초, 최적, 최악 적합 중 하나의 방식으로 할당된다.

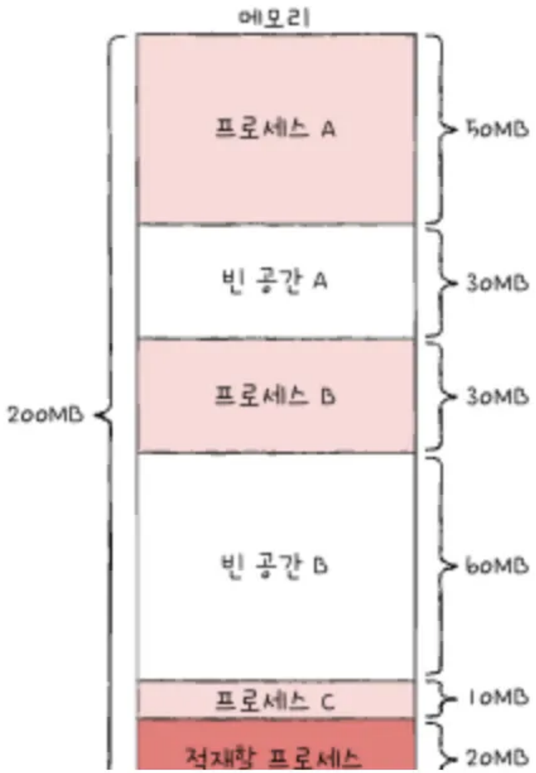
메모리 할당: 최초 적합



빈 공간을 순서대로 검색하다 적재 가능 공간 발견 시 바로 프로세스 배치

검색 최소화, 빠른 할당

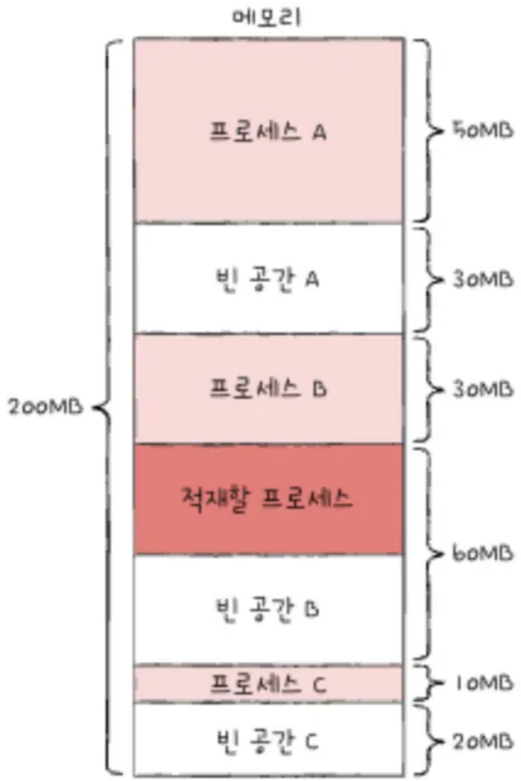
메모리 할당: 최적 적합



빈 공간 모두 검색 후 프로세스가 적재될 수 있는

가장 작은 공간에 프로세스 배치

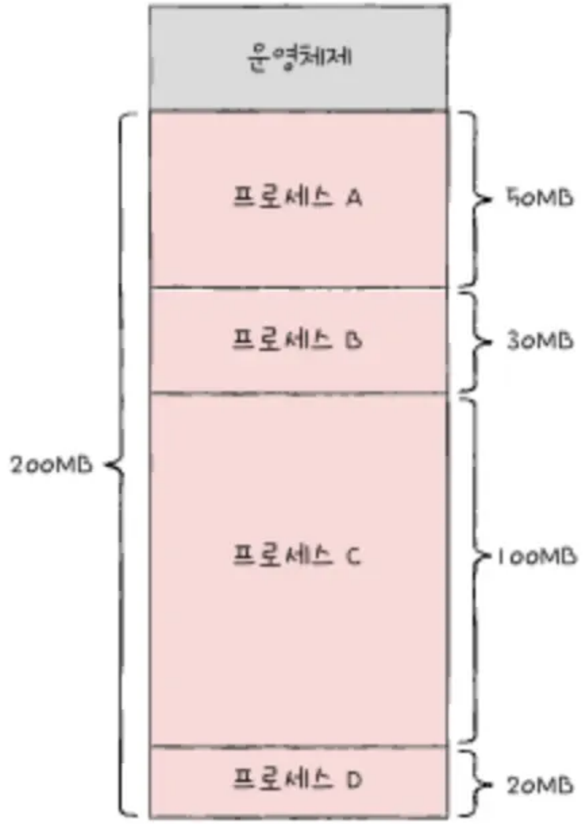
메모리 할당: 최악 적합



빈 공간 모두 검색 후 프로세스가

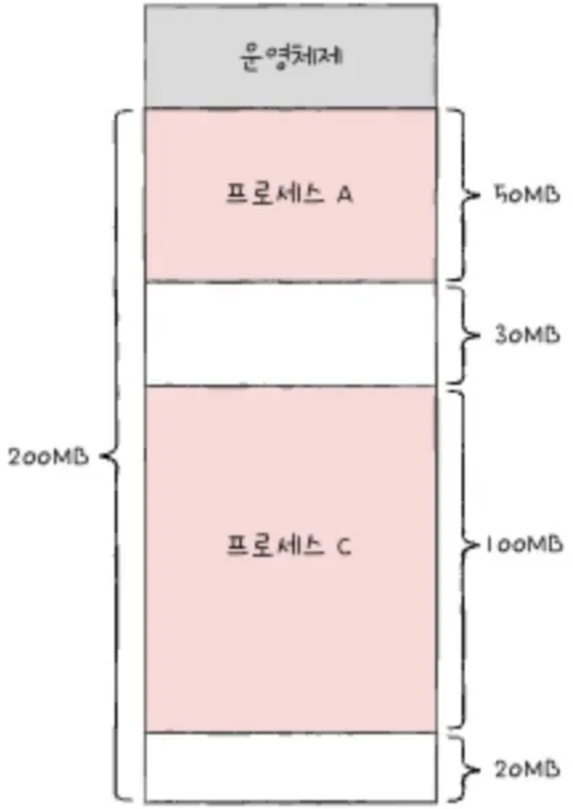
적재될 수 있는 가장 큰 공간에 배치

외부 단편화



프로세스 A, B, C, D를 다음과 같이 딱 맞게 배치했다.

외부 단편화

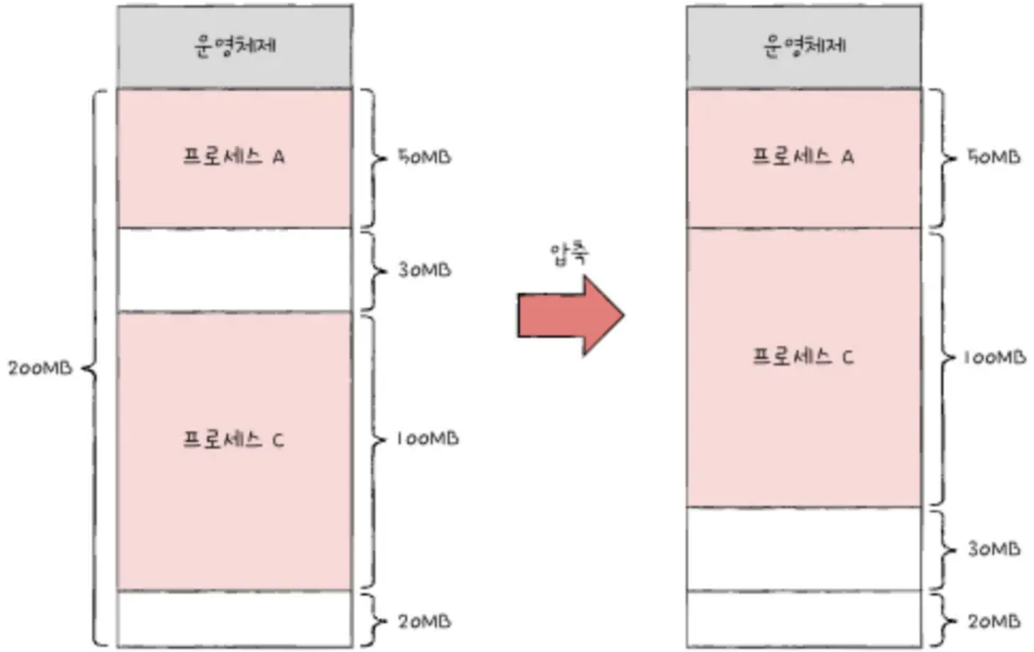


B, D 실행이 끝나면, 빈 공간이 생긴다.

빈 공간 총합은 50MB지만 50MB 크기의 프로세스 적재는 불가능.

이러한 현상을 외부 단편화라고 한다.

외부 단편화 해결 방법: 압축



빈 공간들을 하나로 모으는 방식

하지만 이 과정에서 시스템은 하던 일을 중지해야 한다.

또 메모리 내용을 옮기는 작업은 많은 오버헤드를 야기한다.

또 다른 해결 방법: 페이징