# 페이지 교체 알고리즘

> 페이지 부재 발생 → 새로운 페이지를 할당해야 함 → 현재 할당된 페이지 중 어떤 것 교체할 지 결정하는 방법

- 좀 더 자세하게 생각해보면?

가상 메모리는 요구 페이지 기법을 통해 필요한 페이지만 메모리에 적재하고 사용하지 않는 부분은 그대로 둠

하지만 필요한 페이지만 올려도 메모리는 결국 가득 차게 되고, 올라와있던 페이지가 사용이 다 된 후에도 자리만 차지하고 있을 수 있음

따라서 메모리가 가득 차면, 추가로 페이지를 가져오기 위해서 안쓰는 페이지는 out하고, 해당 공간에 현재 필요한 페이지를 in 시켜야 함

여기서 어떤 페이지를 out 시켜야할 지 정해야 함. (이때 out 되는 페이지를 victim page라고 부름)

기왕이면 수정이 되지 않는 페이지를 선택해야 좋음 (Why? : 만약 수정되면 메인 메모리에서 내보낼 때, 하드디스크에서 또 수정을 진행해야 하므로 시간이 오래 걸림)

> 이와 같은 상황에서 상황에 맞는 페이지 교체를 진행하기 위해 페이지 교체 알고리즘이 존재하는 것!

#### Page Reference String

> CPU는 논리 주소를 통해 특정 주소를 요구함
>
> 메인 메모리에 올라와 있는 주소들은 페이지의 단위로 가져오기 때문에 페이지 번호가 연속되어 나타나게 되면 페이지 결함 발생 X
>
> 따라서 CPU의 주소 요구에 따라 페이지 결함이 일어나지 않는 부분은 생략하여 표시하는 방법이 바로 `Page Reference String`

1. **FIFO 알고리즘**

   > First-in First-out, 메모리에 먼저 올라온 페이지를 먼저 내보내는 알고리즘

   victim page : out 되는 페이지는, 가장 먼저 메모리에 올라온 페이지

   가장 간단한 방법으로, 특히 초기화 코드에서 적절한 방법임

   초기화 코드 : 처음 프로세스 실행될 때 최초 초기화를 시키는 역할만 진행하고 다른 역할은 수행하지 않으므로, 메인 메모리에서 빼도 괜찮음

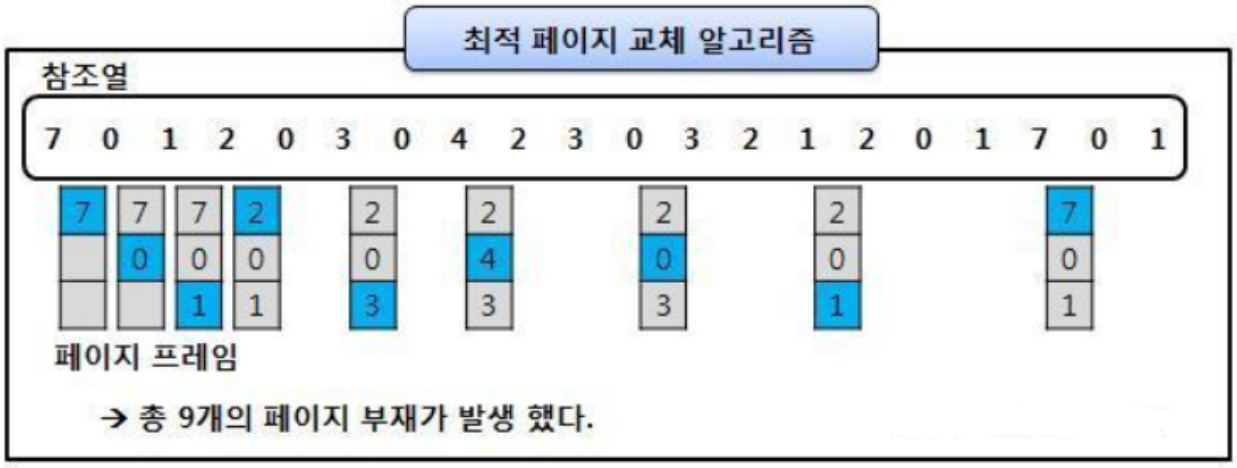
   하지만 처음 프로세스 실행시에는 무조건 필요한 코드이므로, FIFO 알고리즘을 사용하면 초기화를 시켜준 후 가장 먼저 내보내는 것이 가능함


2. **OPT 알고리즘**

> Optimal Page Replacement 알고리즘, 앞으로 가장 사용하지 않을 페이지를 가장 우선적으로 내보냄

FIFO에 비해 페이지 결함의 횟수를 많이 감소시킬 수 있음

하지만, 실질적으로 페이지가 앞으로 잘 사용되지 않을 것이라는 보장이 없기 때문에 수행하기 어려운 알고리즘임




3. **LRU 알고리즘**

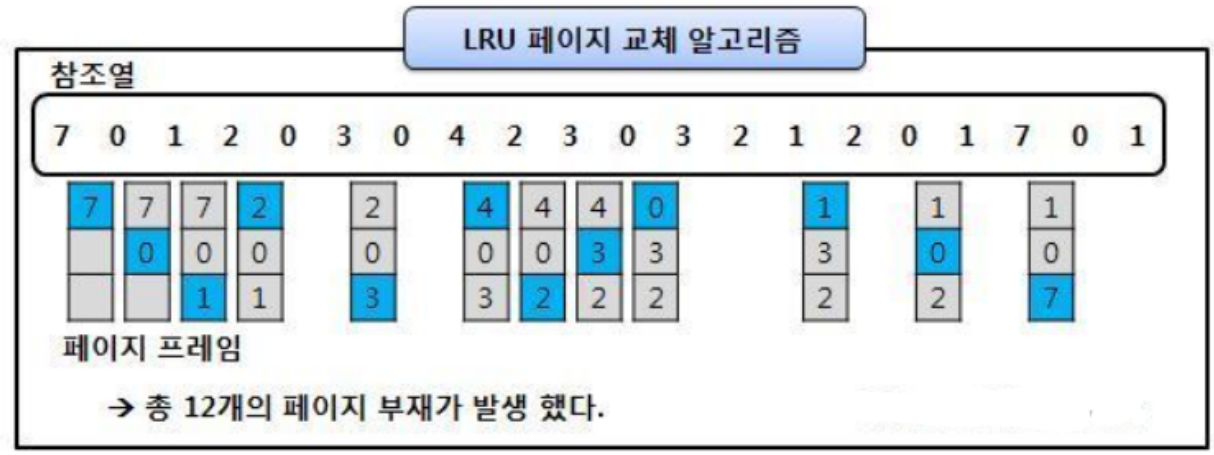
> Least-Recently-Used, 최근에 사용하지 않은 페이지를 가장 먼저 내려보내는 알고리즘

최근에 사용하지 않았으면, 나중에도 사용되지 않을 것이라는 아이디어에서 나옴

OPT의 경우 미래 예측이지만, LRU의 경우는 과거를 보고 판단하므로 실질적으로 사용이 가능한 알고리즘

(실제로도 최근에 사용하지 않은 페이지는 앞으로도 사용하지 않을 확률이 높다)

OPT보다는 페이지 결함이 더 일어날 수 있지만, **실제로 사용할 수 있는 페이지 교체 알고리즘에서는 가장 좋은 방법 중 하나임**




**교체 방식**

- Global 교체

  > 메모리 상의 모든 프로세스 페이지에 대해 교체하는 방식

- Local 교체

  > 메모리 상의 자기 프로세스 페이지에서만 교체하는 방식

다중 프로그래밍의 경우, 메인 메모리에 다양한 프로세스가 동시에 올라올 수 있음

따라서, 다양한 프로세스의 페이지가 메모리에 존재함

페이지 교체 시, 다양한 페이지 교체 알고리즘을 활용해 victim page를 선정하는데, 선정 기준을 Global로 하느냐, Local로 하느냐에 대한 차이

→ 실제로는 전체를 기준으로 페이지를 교체하는 것이 더 효율적이라고 함. 자기 프로세스 페이지에서만 교체를 하면, 교체를 해야할 때 각각 모두 교체를 진행해야 하므로 비효율적