

# Virtual Memory

지난시간 : 물리적인 메모리의 주소 변환은 운영체제가 관여하지 않는다

virtual memory의 경우 관여한다

## Demand Paging

요청이 있을때만 메모리에 페이지를 올려 놓는다

=> I/O 양이 줄어듦

=> Memory 사용량 감소

=> 빠른 응답 시간

### Valid / Invalid bit의 사용

#### ✓ Invalid의 의미

- 사용되지 않는 주소 영역인 경우
- 페이지가 물리적 메모리에 없는 경우

✓ 처음에는 모든 page entry가 invalid로 초기화

✓ address translation 시에 invalid bit이 set되어 있으면

⇒ "page fault".

## Memory에 없는 Page의 Page Table



- page fault : 요청한 페이지가 메모리에 없는 경우

=> cpu가 운영체제에게 넘어감

=> 그럼 운영체제가 fault난 페이지를 메모리에 올림

## Page Fault

- invalid page를 접근하면 MMU가 trap을 발생시킴 (page fault trap)
- Kernel mode로 들어가서 page fault handler가 invoke됨
- 다음과 같은 순서로 page fault를 처리한다
  1. Invalid reference? (eg. bad address, protection violation) ⇒ abort process.
  2. Get an empty page frame. (없으면 뺏어온다: replace)
  3. 해당 페이지를 disk에서 memory로 읽어온다
    1. disk I/O가 끝나기까지 이 프로세스는 CPU를 preempt 당함 (block)
    2. Disk read가 끝나면 page tables entry 기록, valid/invalid bit = "valid"
    3. ready queue에 process를 insert → dispatch later
  4. 이 프로세스가 CPU를 잡고 다시 running
  5. 아까 중단되었던 instruction을 재개

## Page replacement

빈 페이지가 없는 경우에

메모리에서 기존 페이지를 쫓아내고 새로운 메모리로 대체시켜줌 (OS의 일)

Replacement Algorithm

=> 어떤 것을 Disk로 쫓아낼지 정하는 알고리즘

=> page fault rate를 최소화 하는 것이 목표

## Optimal Algorithm

page 순서를 미리 알고 있다고 가정

=> 가장 먼 미래에 참조되는 Page를 쫓아냄

=> 어떤 알고리즘도 이보다 더 적은 replacement를 할 수 없음

=> 그래서 새로만든 알고리즘의 성능 척도 측정용

### Optimal Algorithm

→ MIN (OPT): 가장 먼 미래에 참조되는 page를 replace

→ 4 frames example

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	1	1	1	1	1	1	1	1	1	4	4
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3	3	3	3	3
			4	4	4	5	5	5	5	5	5

6 page faults

→ 미래의 참조를 어떻게 아는가?

- ✓ Offline algorithm

→ 다른 알고리즘의 성능에 대한 upper bound 제공

- ✓ Belady's optimal algorithm, MIN, OPT 등으로 불림

## FIFO Alogorithm

: 먼저 들어온 것을 먼저 내 쫓음

\*\* FiFo Anomally \*\* :

메모리 프레임을 늘렸으나 성능이 별로임

## LRU Algorithm ( Least Recently Used

: 가장 오래전에 참조된 것을 쫓아냄

## LFU Algorithm ( Least Frequently Used

: 참조 횟수가 가장 적은 페이지를 지움

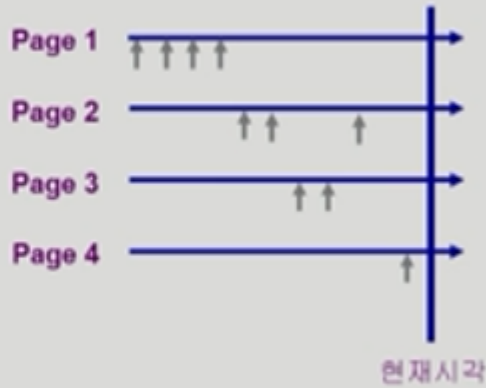
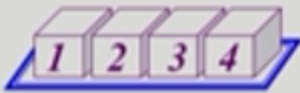
참조횟수가 동일일 경우 임의선정

## LRU와 LFU 알고리즘 예제

### Reference string

현재시각  
↓  
1, 1, 1, 1, 2, 2, 3, 3, 2, 4, 5, ...

### Page Frames



현재시각 5번 page를 보관하기 위해 어떤 page를 삭제해야 하는가?

LRU: 1번 page 삭제  
LFU: 4번 page 삭제

- LRU

### → LRU



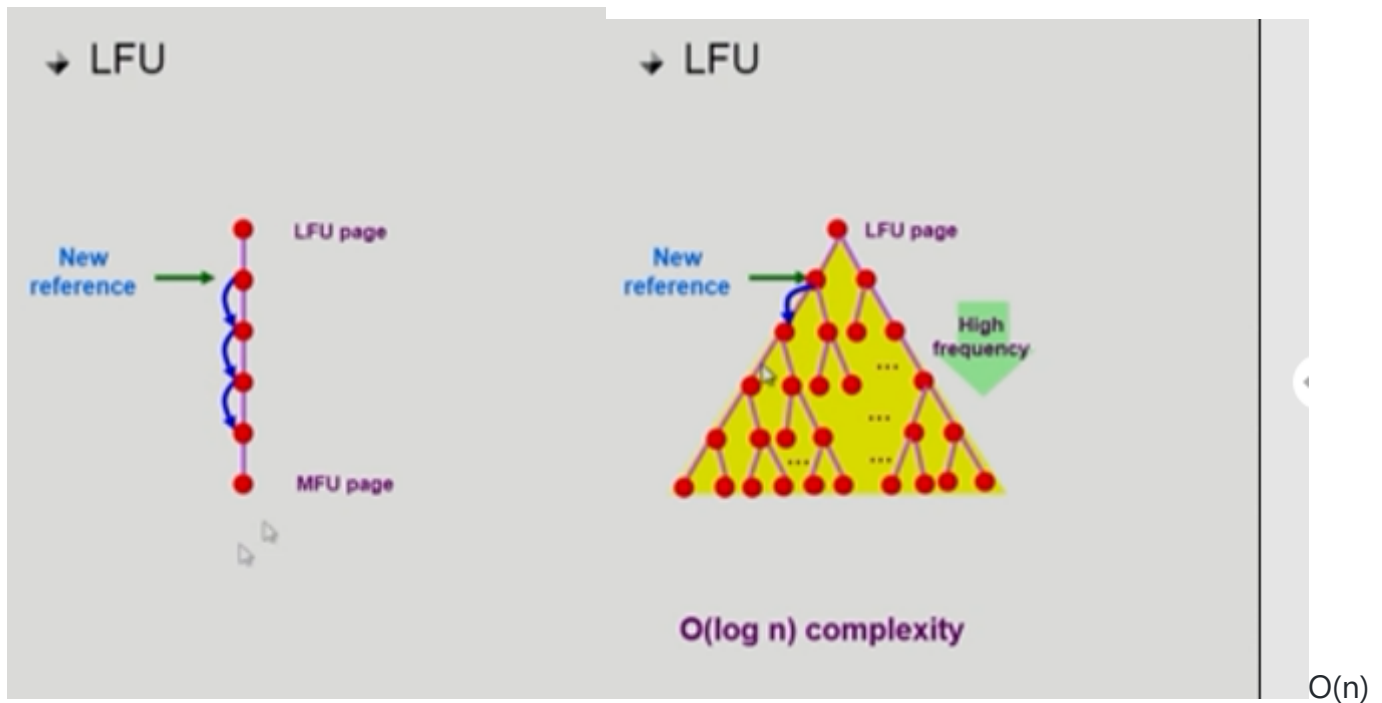
O(1) complexity

어떤 페이지가 참조 될 때마다 맨 아래 메달고

쫓아낼 때는 제일 위에거 쫓아내고

이런 방식으로 구현

- LFU
- 참조횟수 1회 늘어난걸로 바로 다른 페이지들과 비교 가능한 것이 아님
- Heap 이용해서 이진 트리를 구성
- 참조횟수가 바뀔 때 아래위로 비교하고 위치 조정



=  $\log n$

캐쉬

## 다양한 캐싱 환경

### → 캐싱 기법

- ✓ 한정된 빠른 공간(=캐쉬)에 요청된 데이터를 저장해 두었다가 후속 요청시 캐쉬로부터 직접 서비스하는 방식
- ✓ paging system 외에도 cache memory, buffer caching, Web caching 등 다양한 분야에서 사용

### → 캐쉬 운영의 시간 제약

- ✓ 교체 알고리즘에서 삭제할 항목을 결정하는 일에 지나치게 많은 시간이 걸리는 경우 실제 시스템에서 사용할 수 없음
- ✓ Buffer caching이나 Web caching의 경우
  - $O(1)$ 에서  $O(\log n)$  정도까지 허용
- ✓ Paging system인 경우
  - page fault인 경우에만 OS가 관여함
  - 페이지가 이미 메모리에 존재하는 경우 참조시각 등의 정보를 OS가 알 수 없음
  - $O(1)$ 인 LRU의 list 조작조차 불가능

!!! page fault 의 경우에만 운영체제에게 Cpu가 넘어옴

## Clock Algorithm

## Clock Algorithm

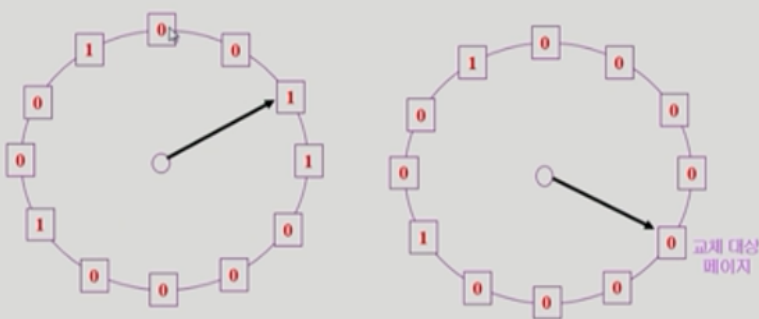
### → Clock algorithm

- ✓ LRU의 근사(approximation) 알고리즘
- ✓ 여러 명칭으로 불림
  - Second chance algorithm
  - NUR (Not Used Recently) 또는 NRU (Not Recently Used)
- ✓ Reference bit을 사용해서 교체 대상 페이지 선정 (circular list)
- ✓ reference bit가 0인 것을 찾을 때까지 포인터를 하나씩 앞으로 이동
- ✓ 포인터 이동하는 중에 reference bit 1은 모두 0으로 바꿈
- ✓ Reference bit이 0인 것을 찾으면 그 페이지를 교체
- ✓ 한 바퀴 되돌아와서도(=second chance) 0이면 그때에는 replace 당함
- ✓ 자주 사용되는 페이지라면 second chance가 올 때 1

### → Clock algorithm의 개선

- ✓ reference bit과 modified bit (dirty bit)을 함께 사용
- ✓ reference bit = 1 : 최근에 참조된 페이지
- ✓ modified bit = 1 : 최근에 변경된 페이지 (I/O를 동반하는 페이지)

## Clock Algorithm



각각의 사각형은 페이지 프레임

Reference bit : 페이지가 참조될 때 1로 체크해줌, 하드웨어의 일

Reference bit = 1 이라는 뜻 == 최근에 참조됨

modified bit :

write 참조 일 때 내용이 수정 되었다는 뜻이므로 Disk에 변경사항 써주고 쫓아냄

## Page Frame의 Allocation

## Page Frame의 Allocation

- Allocation problem: 각 process에 얼마만큼의 page frame을 할당할 것인가?
- Allocation의 필요성
  - ✓ 메모리 참조 명령어 수행시 명령어, 데이터 등 여러 페이지 동시 참조
    - 명령어 수행을 위해 최소한 할당되어야 하는 frame의 수가 있음
  - ✓ Loop를 구성하는 page들은 한꺼번에 allocate 되는 것이 유리함
    - 최소한의 allocation이 없으면 매 loop 마다 page fault
- Allocation Scheme
  - ✓ **Equal allocation**: 모든 프로세스에 똑 같은 갯수 할당
  - ✓ **Proportional allocation**: 프로세스 크기에 비례하여 할당
  - ✓ **Priority allocation**: 프로세스의 priority에 따라 다르게 할당

- Equal allocation : 모두 균등 할당
- Proportional allocation : 프로그램 크기에 비례해서 페이지 할당

## Global vs. Local Replacement

- **Global replacement**
  - ✓ Replace 시 다른 process에 할당된 frame을 빼앗아 올 수 있다
  - ✓ Process별 할당량을 조절하는 또 다른 방법임
  - ✓ FIFO, LRU, LFU 등의 알고리즘을 global replacement로 사용시에 해당
  - ✓ Working set, PFF 알고리즘 사용
- **Local replacement**
  - ✓ 자신에게 할당된 frame 내에서만 replacement
  - ✓ FIFO, LRU, LFU 등의 알고리즘을 process 별로 운영시

Global replacement



replacement 알고리즘이 알아서 할당 효과를 내게 함

Local replacement

프로그램들에게 미리 메모리를 할당함

새로운 페이지가 필요할 때 자신에게 할당된 페이지를 쫓아냄