

# OS\_페이지 교체

## 메모리 할당 예제

### 예제 1


연속 메모리 할당 시스템

빈 공간 홀 100KB, 500KB, 600KB, 300KB, 200KB

212KB, 417KB, 112KB, 426KB의 크기의 프로세스를 순서대로 배치

- (a) first-fit, best-fit, worst-fit 중에서 외부 단편화 최소화 할 수 있는 것은?

hole



· First-fit	212	417	112	$\Rightarrow 100+200+(500-212)+(600-417)+(300-112) = 959KB$
· Best-fit	417	426	212	112 $\Rightarrow 100+(500-417)+(600-426)+(300-212)+(200-112) = 533KB$
· Worst-fit	417	212	112	$\Rightarrow 100+200+(600-417)+(300-212)+(200-112) = 959KB$


$\therefore$  Best-fit, 533KB

- (b) 페이징 이용하는 시스템 홀 크기와 프로세스 크기 및 배치 순서는 위와 동일, 내부 단편화로 인해 낭비되어지는 메모리 크기는?

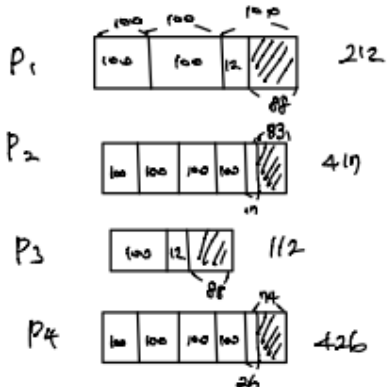
b) 페이징 이용하는 시스템. 홀 크기와 프로세스 크기는 위와 동일

내부 단편화로 인해 낭비되어지는 메모리 크기는? 페이지 크기 = 100KB

hole



$\therefore$  내부 단편화 크기:  $88+83+88+74 = 333KB$



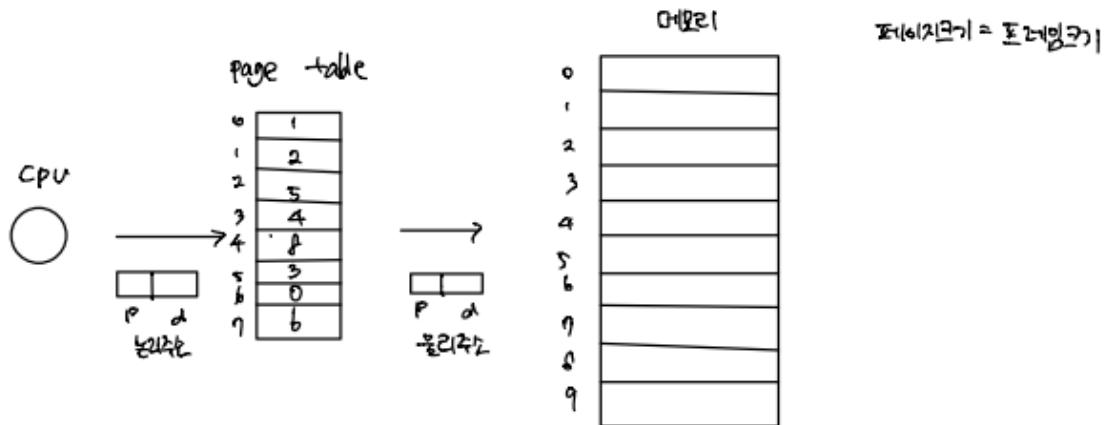
## 예제 2

메모리 관리 페이징 이용하는 시스템

한 페이지의 크기는 1KB

테이블 내용 순서대로 1 2 5 4 8 3 0 6

- (a) 논리주소 3000번지는 물리주소 몇번지인가?



$$\begin{aligned}
 (a) \quad 3000 &= 1011\ 1011\ 1000_2 \\
 \text{페이지크기} &= 1\text{KB} = 2^{10} \text{ bytes} \Rightarrow d = 10 \text{ bits} \\
 \Rightarrow p &= 10_2 = 2, \quad d = 11\ 1011\ 1000_2 \\
 &\quad (\text{논리주소, page table 인덱스}) \\
 &\quad \downarrow \\
 &\quad (\text{물리주소, Frame number} = 5) \\
 p &= 5, \quad d = 11\ 1011\ 1000_2 \quad (d \text{는 그대로}) \\
 &= 101_2 \\
 \therefore \text{물리주소} &= 101\ 11\ 1011\ 1000_2 \quad (= 1738_{10})
 \end{aligned}$$

- (b) 물리주소 1A53(16진수)는 논리주소 몇번지인가?

$$\begin{aligned}
 (b) \quad 1A53_{16} &= 11010\ 01010011_2 \\
 \text{물리주소} : p &= 110_2 = 6 \quad d = 1001010011_2 \\
 &= 6 \quad (\text{프레임번호}) \\
 &\quad \downarrow \\
 \text{논리주소} : p &= 7 \quad (\text{페이지번호인덱스}), \quad d = 1001010011_2 \quad (\text{그대로}) \\
 &= 111_2 \\
 \therefore \text{논리주소} &= 111\ 1001010011_2 = 1E53_{16}
 \end{aligned}$$

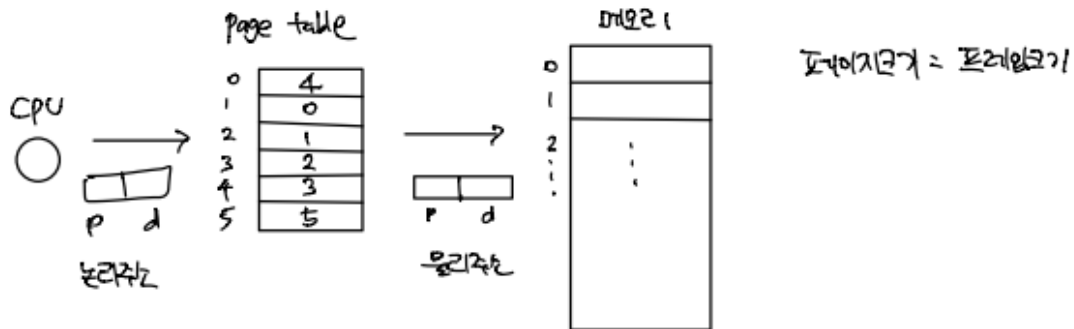
### 예제 3

메모리 관리 기법 페이징 사용 시스템

한 메모리 크기 512바이트

페이지 테이블 내용 순서대로 4 0 1 2 3 5

- (a) 논리주소 0X123 일 때 물리주소는?



$$(a) \quad 0X123_{16} = 100100011_2$$

$$\text{페이지크기} = 512 \text{ bytes} = 2^9 \text{ bytes} \Rightarrow d = 9 \text{ bits}$$

$$p = 0, \quad d = 100100011_2$$

(논리주소, 페이지테이블 인덱스)

↓

$$p = 4 = 100_2, \quad d = 100100011_2 \text{ (그대로)}$$

(물리주소, 프레임번호)

$$\therefore \text{물리주소} = 100100100011_2 = 0X923 \text{ (16진수)}$$

- (b) 물리주소 1500 일 때 논리주소는?

$$(b) \quad \text{물리주소 } 1500 = 10111011100_2$$

$$\text{페이지크기} = 2^9 \text{ bytes} \Rightarrow d = 9 \text{ bits}$$

$$p = 10_2 = 2, \quad d = 111011100_2$$

(물리주소, 프레임번호)

↓

$$p = 3 = 11_2 \text{ (논리주소, 인덱스)}, \quad d = 111011100_2$$

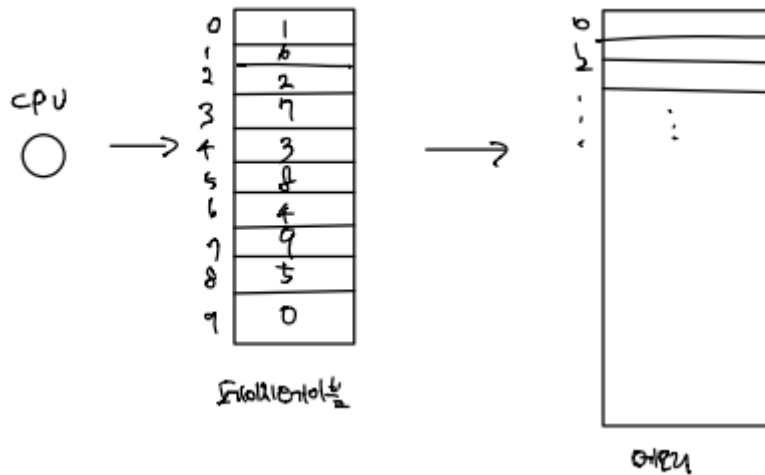
$$\therefore \text{논리주소} = 11111011100_2 = 0X7DC \text{ (16진수)}$$

## 예제 4

페이지 크기 16 바이트인 페이지징 시스템

페이지 테이블 내용 순서대로 1 6 2 7 3 8 4 9 5 0

- (a) 논리주소 50 일 때 물리주소는?



$$(a) \ 50 = 110010_2$$

$$\text{페이지 크기 16바이트} = 2^4 \rightarrow d = 4 \text{bits}$$

$$p = 11_2 = 3 \text{ (논리주소, 인덱스)}, d = 0010_2$$

↓

$$p = 7 = 111_2 \text{ (물리주소, 프레임번호)}, d = 0010_2$$

$$\therefore \text{물리주소} = 1110010_2$$

- (b) 물리주소  $1011001_2$  일 때 논리주소는?

$$(b) \ 1011001_2 \Rightarrow p = 101_2 = 5 \text{ (물리주소, 프레임번호)}, d = 1001_2$$

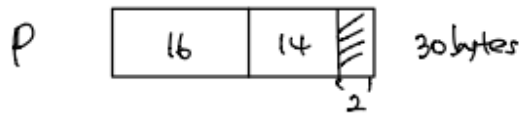
↓

$$p = 8 = 1000_2 \text{ (논리주소, 인덱스)}, d = 1001_2$$

$$\therefore \text{논리주소} = 10001001_2$$

- (c) 30 바이트 크기 프로세스 배치하려고 한다. 내부 단편화로 인한 메모리 손실 크기는?

(C) 페이지 크기 = 프레임 크기 = 16바이트

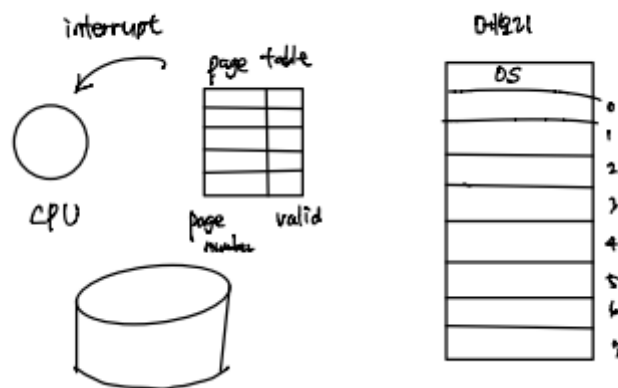


∴ 2byte 낭비

## 페이지 교체

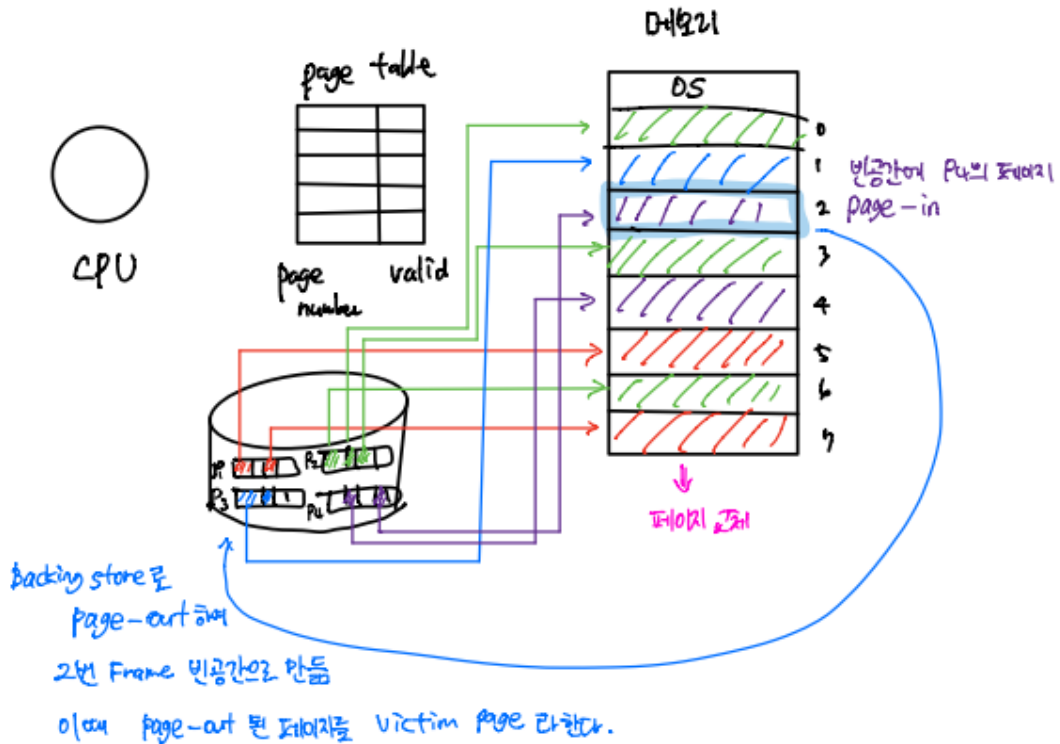
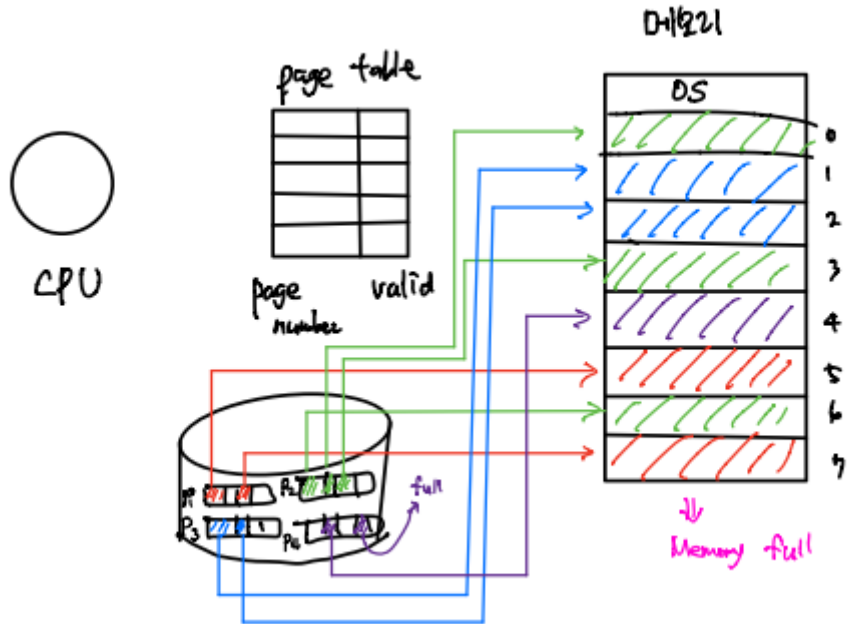
### Demand Paging

- 요구되어지는 페이지만 backing store에서 가져온다
- 프로그램 실행 계속에 따라 요구 페이지가 늘어남
- 언젠가는 메모리가 가득차게 된다.



- CPU에서 주소냈는데 valid 비트가 0이면(메모리에 로드 X) Interrupt를 CPU에 보냄
- Interrupt 신호 받으면 OS에서 Interrupt 처리 루틴 수행 => 하드디스크에서 해당 페이지를 찾아 메모리에 로드
- page fault 일어난 page table의 page number를 방금 로드한 메모리 주소로 바꾸고 valid 비트를 1로 바꿈

### Memory Full



- 메모리가 가득 차면 추가로 페이지를 가져오기 위해
- 어떤 페이지는 backing store로 몰아내고 (page-out)
- 그 빈 공간으로 페이지를 가져온다 (page-in)
- 이 때 page-out된 페이지를 victim page라 한다

## Victim Page

어떤 페이지를 몰아낼 것인가?

	page num	valid	modified	
1	3	1	1	→ 실행하면서 수정된 page
2	4	1	0	→ 수정되지 않은 page
3	1	1	1	
4	6	1	0	
5	2	1	0	
6	5	1	1	

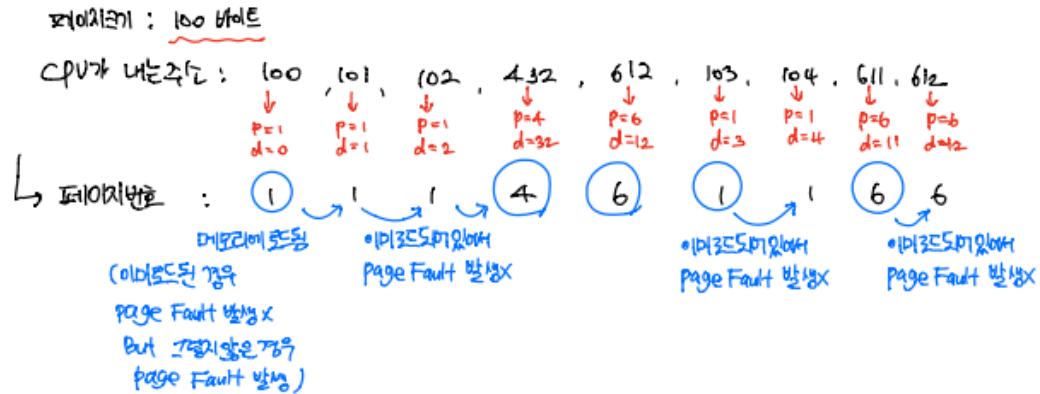
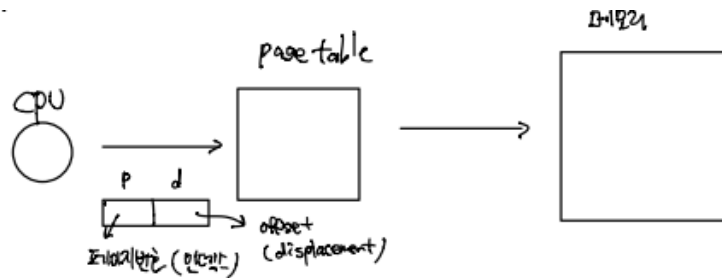
- I/O 시간 절약을 위해
- 이왕이면 modify 되지 않은 페이지를 victim으로 선택
  - 실행하면서 내용이 바뀐 페이지는 하드디스크로 돌려보낼 때 write 해줘야함 => 하드디스크에 write 하는거라 시간 오래걸림
  - 메모리에서 몰아낼 때 하드디스크에 write 안해도 됨
- 방법
  - modified bit (= **dirty bit**) => valid 비트처럼 1비트 추가 한 것 => 수정되었다면 1, 아니면 0

## 여러 페이지 중에서 무엇을 Victim으로?

- Random
  - modified bit 1인 page 중에서 랜덤으로 victim 선택, 성능↓
- First-in-First-out(FIFO)
  - modified bit 1인 page 중에서 메모리에 먼저 로드된 page를 victim으로 선택
- 그 외, 페이지 교체 알고리즘
  - OPT(Optimal)
  - LRU(Least-Recently-Used)

## Page reference string

- 페이지 참조 열
- 예제



○ : page Fault 발생할 수 있음.

⇒ 페이지 참조 열 : 1 4 6 1 6

- CPU가 내는 주소: 100 101 102 432 612 103 104 611 612
- 페이지 크기: 100 bytes => 최대 offset: 99
- 페이지 번호: 1 1 1 4 6 1 1 6 6 (물리주소X, 페이지 테이블 인덱스)
- Page reference string: 1 4 6 1 6

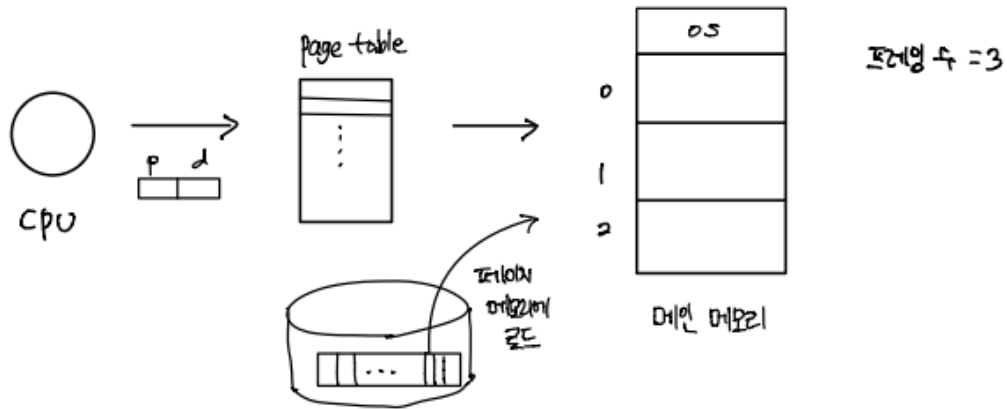
## 페이지 교체 알고리즘

### FIFO (First In First Out)

메모리에 먼저 로드된 페이지를 먼저 내보낸다(victim으로 선택한다)

- Simplest
  - idea: 초기화 코드는 더 이상 사용되지 않을 것
- 예제
  - 페이지 참조열 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
  - number of frames = 3 (메모리 프레임 3개)
  - 15 page faults





- 처음 power on  $\Rightarrow$  빈 메모리 상태.

① 페이지 참조번호 = 7  $\Rightarrow$  빈 메모리 상태라 page fault 발생  $\Rightarrow$  하드디스크에서 page 7 찾아서 (page fault count = 1) 메모리에 로드

7

② 페이지 참조번호 = 0  $\Rightarrow$  메모리에 있음, page fault 발생 (page fault count = 2)

7
0

③ 페이지 참조번호 = 1  $\Rightarrow$  메모리에 있음, PF 발생 (PF count = 3)

7
0
1

④ 페이지 참조번호 = 2  $\Rightarrow$  메모리에 있음, PF 발생 (PF count = 4), 메모리에서 7번과 2번을 (FIFO)

2
0
1

page	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
Frame (메모리)	7	0	1	2	2	3	0	4	2	3	0	0	0	1	2	2	2	7	0	1
	7	0	1	1	2	3	0	4	2	3	3	3	3	0	1	1	1	2	7	0
	7	0	0	1	2	3	0	4	2	2	2	2	3	0	0	0	0	1	2	7
PF 발생	0	0	0	0	X	0	0	0	0	0	0	X	X	0	0	X	X	0	0	0

메모리에 0 존재

메모리에 3 존재

메모리에 2 존재

메모리에 0 존재

메모리에 1 존재

$\therefore$  PF 발생: 15회

$\hookrightarrow$  위 표는 새로 보시면 됩니다! ex) 페이지 참조번호 1 일 때 이전 메모리 (2.1.0)은 가장 처음로드된 7이 빠지고 2가 새로 메모리에 로드된 것을 뜻합니다. 페이지 교체 발생했으므로 PF 발생 0

실제 프레임은 위 표와 같이 로드된 것은 아래쪽으로 밀리지 않습니다!!

가장 먼저로드된 페이지 번호를 찾기 편하도록 표현하기 위해 큐처럼 나타낸 것입니다.

### • Belady's Anomaly

- 프레임 수 (= 메모리 용량) 증가에 PF(page fault) 회수 증가?

