

∴ 09 장부터 메모리 관리 기법에 Paging 기법을
사용한다고 가정

실제로 사용

→ demand paging 기법

Demand Paging

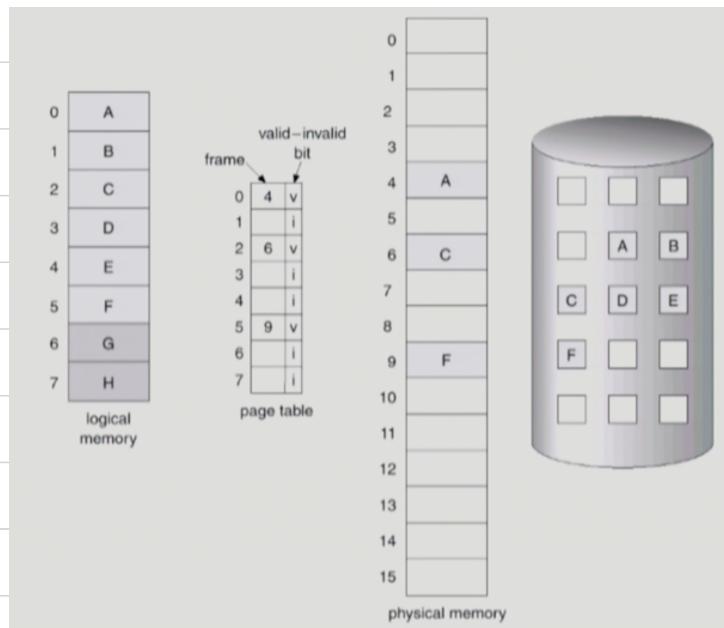
- 실제로 필요할 때 page를 메모리에 옮기는 것

- I/O 양의 감소
- Memory 사용량 감소
- 빠른 응답 시간
- 더 많은 사용자 수용

• Valid/Invalid bit의 사용

- Invalid bit 의 용도
 - 사용되지 않는 주소영역인 경우
 - 페이지가 물리적 메모리에 있는 경우
- 처음에는 모든 Page entry가 invalid로 초기화
- Address translation 시에 invalid bit이 set 되어 있으면
⇒ "Page fault" → 파동으로 CPU 사용이 운영체계로 놓여가며 trap됩니다.

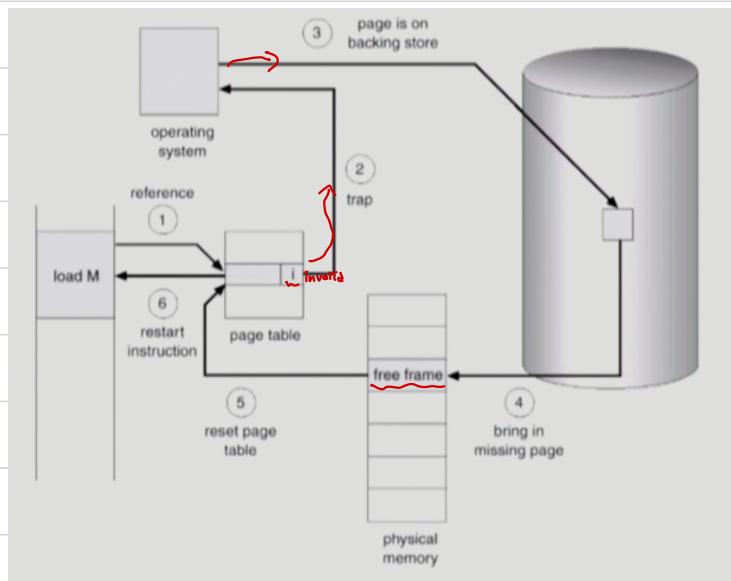
Memory에 있는 Page의 Page Table



Page fault

- invalid page를 접근하면 MMU가 trap을 발생시킴 (page fault trap)
- Kernel mode로 돌아가서 page fault handler가 invoke됨
- 다음과 같은 순서로 Page fault를 처리한다.
 1. Invalid reference? (e.g. bad address, protection violation) => abort process
 2. Get an empty page frame. (없으면 뺏어온다: replace)
 3. 해당 페이지는 Disk에서 memory로 읽어온다.
 1. Disk I/O가 끝나기 까지 이 프로세스는 CPU를 preempt 당함 (block)
 2. Disk read가 끝나면 page table entry 기록, valid/invalid bit = "valid"
 3. ready queue의 process를 insert → dispatch later
 4. 이 프로세스가 CPU를 잡고 다시 running
 5. 아까 끊은 되었던 instruction을 재개

Steps in Handling a Page Fault



Performance of Demand Paging

• Page Fault Rate $0 \leq p \leq 1.0$

- v if $p=0$ no page faults
- v if $p=1$, every reference is a fault

• Effective Access Time

$$= (1-p) \times \text{memory access}$$

$$\begin{aligned} &+ p (\text{OS \& HW Page fault overhead} \\ &\quad + [\text{swap page out if needed}]) \\ &\quad + \text{swap page in} \\ &\quad + \text{OS \& HW restart overhead}) \end{aligned}$$

Page fault가 발생했을 때
공장의 older head가 크게 낙방

Free frame 없는 경우

• Page replacement

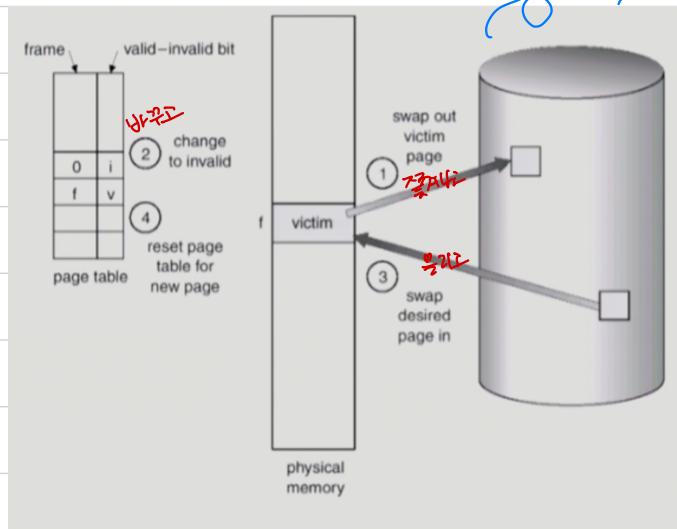
- v 어떤 페이지를 빼앗아 옮지 결정해야 함
- v 끝까지 사용되지 않을 page를 뽑아내는 것이 좋음
- v 동일한 페이지가 여러 번 메모리에서 쫓겨 났다가 다시 들어올 수 있음

• Replacement Algorithm

- v page fault rate를 최소화 하는 것이 목표
- v 알고리즘의 평가
 - 주어진 page reference string에 대해 page fault를 얼마나 내는지 조사.
- v reference string의 예
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

시작 히트비 퍼포먼스
Page가 충돌된 순서

Page Replacement



요약할은 운영체제가
하게 되는 것

Optimal Algorithm

Page fault을 가장 적게 하는 알고리즘

미래에 참조될 것들을 미리 알면 가능함. ⇒ 실제로 사용할 수 있음

- MIN(OPT) : 가장 먼 미래에 참조되는 Page를 replace
- 4 frames example

1,	2,	3,	4,	1,	2,	5,	1,	2,	3,	4,	5
→ 미리예측 정적 참조											
1	1	1	1	1	1	1	1	1	1	4	4
2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	5	5	5	5	5	5
fault				fault				fault			

- 미래의 참조를 어떻게 아는 것인가?

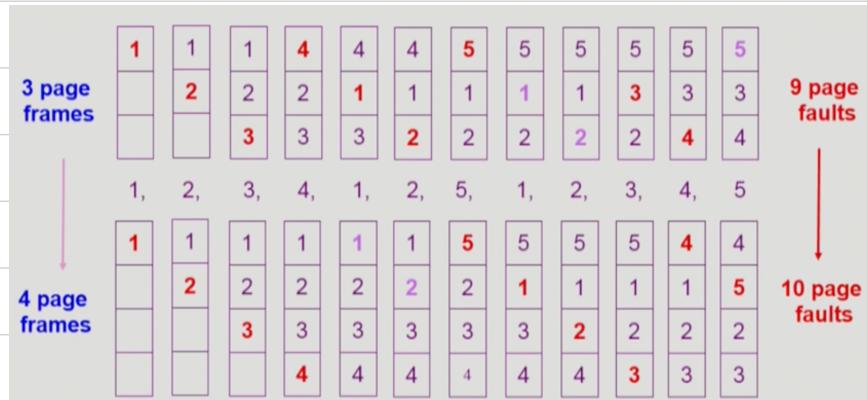
▼ off line algorithm

- 다른 알고리즘의 성능에 대한 upper bound 제공

▼ Belady's optimal algorithm, MIN, OPT 등으로 불립

FIFO (First In First Out) Algorithm

- FIFO: 먼저 들어온 걸 먼저 내쫓음.

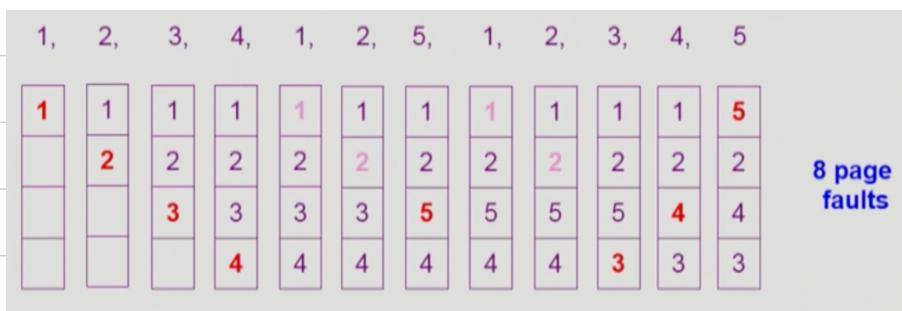


- FIFO Anomaly (Belady's Anomaly)

more frames $\not\Rightarrow$ less page faults \rightarrow 가역한 현상... page fault가 늘어나..

LRU (Least Recently Used) Algorithm

- LRU: 가장 오래 전에 참조된 것을 지움



LFC (Least Frequently Used) Algorithm

- LFU: 참조 횟수(reference count)가 가장 적은 페이지를 지움

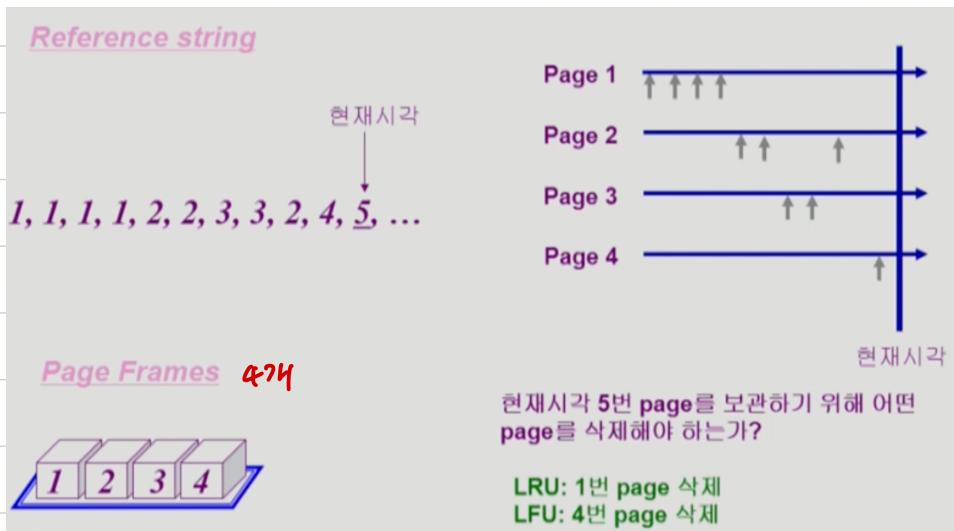
▼ 최저 참조 횟수인 페이지가 여럿 있는 경우

- LFU 알고리즘 자체에서는 어떤 Page를 임의로 선택한다.
- 성능 향상을 위해 가장 오래된에 참조된 Page를 지우게 구현할 수도 있다.

▼ 장단점

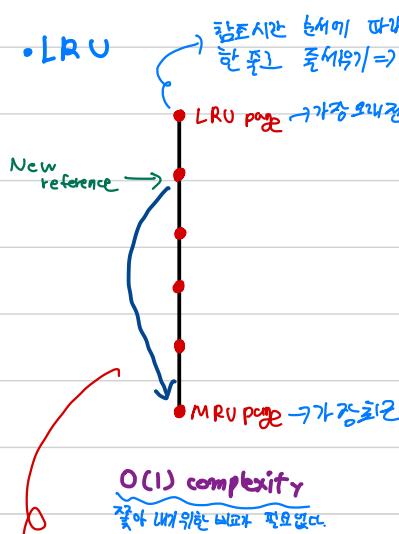
- LRU처럼 적힌 빈도 사용한 보는게 아니라 경기적인 시장 규칙 보기 때문에 Page의 인기도를 끌 더 정확히 반영할 수 있음.
- 참조 시점의 최근성을 반영하지 못함.
- LFU 보다 구현이 복잡함.

LRU와 LFU 알고리즘 예제

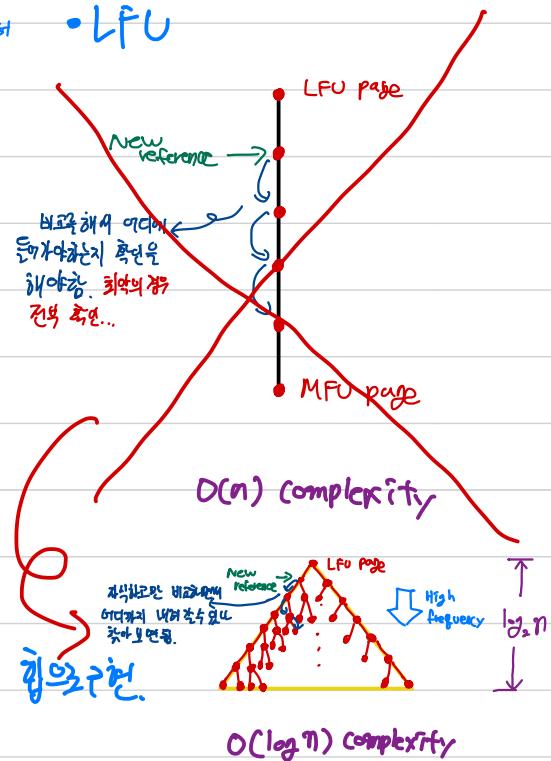


LRU와 LFU 알고리즘의 구현

• LRU



• LFU



다양한 캐시 환경

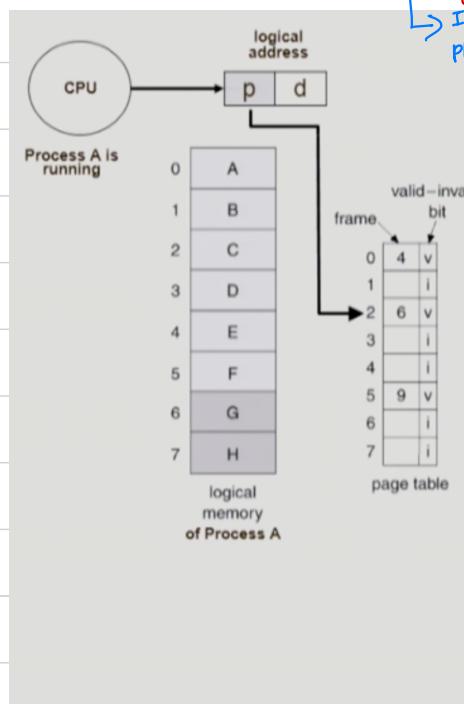
• 캐시 기법

- 한정된 빠른 공간 (=캐시)에 요청된 데이터를 저장해 두었다가 후속 요청시 캐시로부터 적령 서비스 하는 방식
- Paging System 일부도 cache memory, buffer caching, Web caching 등 다양한 분야에서 사용

• 캐시 운영의 시간 제약

- 교체 알고리즘끼리 삭제할 항목을 결정하는 일에 지나치게 많은 시간이 걸리는 경우 실제 시스템에 사용할 수 없음.
- Buffer caching이나 Web caching의 경우
 - O(1)에서 O(log_n) 정도 까지 사용
- Paging System인 경우
 - Page fault인 경우에만 OS가 관여함.
 - 페이지가 이미 메모리에 존재하는 경우 참조시간 등의 정보로 OS가 알 수 없음
 - O(1)인 LRU의 list 조작과 불가능

Paging System에서 LRU, LFU 가능한가?



(Page fault)
I/O로 인하여 CPU 사용이 운영체제로 넘어 왔을 때 운영체제는
Physical memory에 올라가 있는 page 들의 LRU, LFU를 알 수 있나?

⇒ 알 수 없다. Page fault가 발생할 때만
운영체제에게 CPU 사용 경향이 전기때문에 운영체제가
페이지 티의 연산 정보를 알고 있을 수 없다.
∴ 다른 Paging System이 이러한 알고리즘 사용의
부적절하다.



physical memory

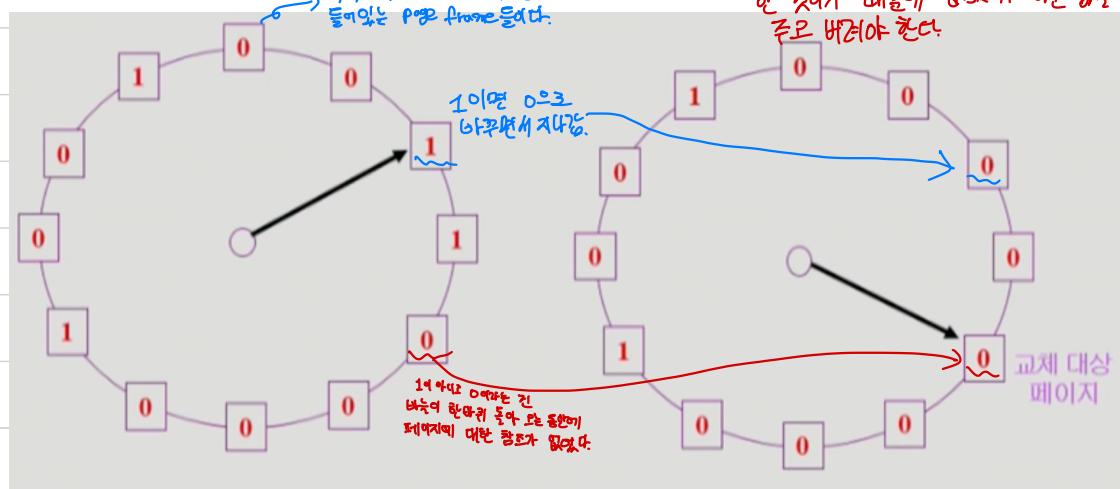
Clock Algorithm

• Clock algorithm

- LRU의 근사(approximation) 알고리즘
- 여러 명칭으로 불림
 - Second Chance algorithm
 - NUR(Not Used Recently) 또는 NRUC(Not Recently Used)
- reference bit을 사용해서 교체 대상 페이지 순환(circular list)
 - reference bit가 0인 것을 찾을 때까지 포인터를 하나씩 앞으로 이동
 - 포인터 이용하는 품에 reference bit 1은 모두 0으로 바꿈
 - Reference bit가 0인 것을 찾으면 그 페이지를 교체
 - 한 바퀴 되돌아 와서도(=Second chance) 0이면 그 때에는 replace로 당함
 - 자주 사용되는 페이지라면 Second chance가 올때 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(디오를 통한하는 페이지)



Page Frame의 Allocation

- Allocation problem: 각 process에 얼마만큼의 page frame을 할당 할 것인가?

- Allocation의 필요성

- 메모리 참조 명령어 수행 시 명령어, 데이터 등 여러 페이지 동시참조
 - 명령어 수행을 위해 최소한 할당되어야 하는 frame의 수가 있음
- Loop를 구성하는 page들은 한꺼번에 allocate 되는 것의 유익함.
- 최소한의 allocation이 없으면 매 loop마다 page fault

- Allocation Scheme

- Equal allocation: 모든 프로세스에 똑같은 갯수 할당

- Proportional allocation: 프로세스 크기에 비례하여 할당

- Priority allocation: 프로세스의 priority에 따라 더 크게 할당

Global vs. Local Replacement

- Global replacement

- Replace 시 다른 process에 할당된 frame을 빼앗아 올 수 있다.
- Process 별 할당량을 조절하는 또 다른 방법임
- FIFO, LRU, LFU 등의 알고리즘을 global replacement으로 사용하기 어렵다
- Working set, PFF 알고리즘 사용

- Local replacement

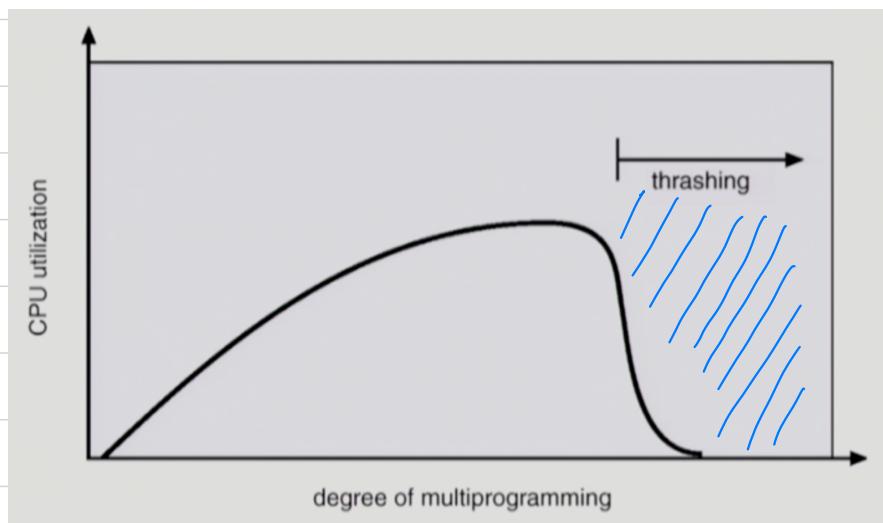
- 자신에게 할당 된 frame 내에서만 replacement
- FIFO, LRU, LFU 등의 알고리즘을 process 별로 운영시

Thrashing

• Thrashing → Page fault가 비일기 발생

- 프로세스의 원활한 수행에 필요한 최소한의 Page frame 수를 할당 받지 못한 경우 발생
- Page fault rate이 매우 높아짐
- CPU utilization이 높아짐 → CPU가 놓고있나?
- OS는 MPD(Multiprogramming degree)를 높여야 한다고 판단
- 또 다른 프로세스가 시스템에 추가됨 (higher MPD)
- 프로세스 당 할당된 frame의 수가 더욱 감소
- 프로세스는 page의 swap in / swap out으로 매우 바쁜
- 대부분의 시간에 CPU는 한가롭
- low throughput

Thrashing Diagram



Working - Set Model

• Locality of reference

- ✓ 프로세스는 특정 시간 동안 일정 장소만을 집중적으로 참조한다.
- ✓ 집중적으로 참조되는 해당 page들의 집합을 locality set 이라 함

• Working - Set Model

- ✓ Locality에 기반하여 프로세스가 일정 시간동안 원활하게 수행되기 위해 한꺼번에 메모리에 올라와 있어야 하는 page들의 집합을 Working set 이라 정의함
- ✓ Working set 모델에서는 process의 working set 진입과 미(영)이 올라와 있으면 수행되고 그렇지 않을 경우 모든 frame을 뺌 후 Swap out (suspend)
↳ 5개 필요한데 3개를 가능하다면 그냥 다 뺌
- ✓ Thrashing을 방지함
- ✓ Multiprogramming degree를 설정함.

Working - Set Algorithm

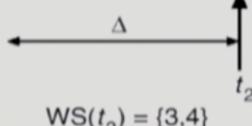
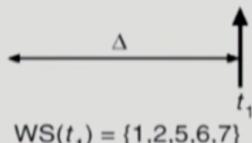
• Working set의 결정

- ✓ Working set window를 통해 알파벳
- ✓ Window size 가 4인 경우
 - 시각 t_i 에서의 Working set $WS(t_i)$
 - Time interval $[t_i : t_i + \Delta]$ 사이에 참조된 서로 다른 페이지들의 집합
 - Working set에 속한 page는 메모리에 유지, 속하지 않은 것은 버림
(즉, 참조된 후 Δ 시간동안 해당 page를 메모리에 유지한 후 버림)

과거를 통해 추정하는 것

page reference table

... 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 ...



Working - Set Algorithm

• Working - Set Algorithm

- ✓ process 틀의 working set size 를 page frame의 수보다 큰 경우
 - 일부 process를 swap out 시켜 남은 process의 working set을 우선적으로 통조理 처리 했다(MPD를 줄임)
- ✓ Working set을 다 할당 하려고 page frame의 남는 경우
 - Swap out 되었던 프로세스에게 Working set을 할당(MPD를 키움)

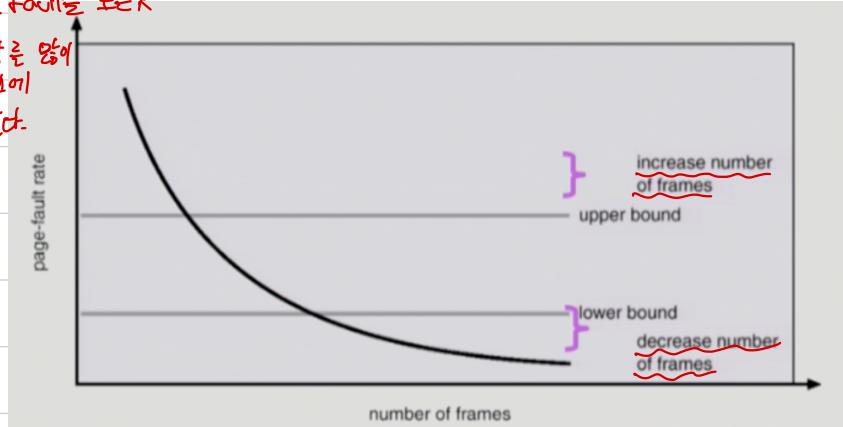
• Window Size 4

- ✓ Working set을 제대로 통조理하기 위해선 window size를 잘 결정해야 함
- ✓ Δ값이 너무 작으면 locality set을 모두 사용하지 못할 우려
- ✓ Δ값이 크면 여러 구조의 locality set 사용
- ✓ Δ값이 00이면 전체 프로그램을 구성하는 page를 working set으로 간주

PFFC (Page - Fault Frequency) Scheme

↳ 직접 page fault를 보는 것

=> page fault를 많이
일으키는 프로그램에
frame을 더 준다.



• Page-fault rate의 상한값과 하한값을 듣다.

- ✓ Page-fault rate의 상한값을 넘으면 frame을 더 할당한다.
- ✓ Page-fault rate의 하한값 이하이면 할당 frame 수를 줄인다.
- 빈 frame이 없으면 일부 프로세스를 swap out

Page Size의 결정

• Page size를 감소시키면 $\nearrow 4KB$

- ✓ 페이지 수 증가
- ✓ 페이지(테이블 크기) 증가
- ✓ Internal fragmentation 감소
- ✓ Disk Transfer의 효율성 감소
 - Seek/Rotation vs. transfer
- ✓ 필요한 정보만 메모리에 올라와 메모리 이용이 효율적
- Locality의 활용 측면에서는 좋지 않음.

• Trend

- ✓ Larger page size

→ 4KB보다 큰...