쉽게 배우는

운영체제

2판

Chapter 09 가상 메모리 관리

차례

학습목표

- 요구 페이징의 개념을 이해하고 구현 방법을 알아본다.
- 페이지 교체 알고리즘의 종류와 각각의 동작 원리를 알아본다.
- 스레싱의 개념을 이해하고 스레싱을 완화하는 프레임 할당 방법을 알아본다.

1-1 요구 페이징의 개요

■ 프로세스의 일부만 메모리로 가져오는 이유

- 메모리를 효율적으로 관리하기 위해: 메모리가 꽉 차면 관리하기 어려우므로 가급적 적은 양의 프로세스만 유지
- 응답 속도를 향상하기 위해: 용량이 큰 프로세스를 전부 메모리로 가져와 실행하면 응답이 늦어질 수 있으므로 필요한 모듈만 올려 실행

■ 포토샵 예

- 메모리에는 포토샵의 메인 프로그램만 올리고 필터는 사용자가 필요로 할 때마다 메모리로 가져오는 것이 효율적

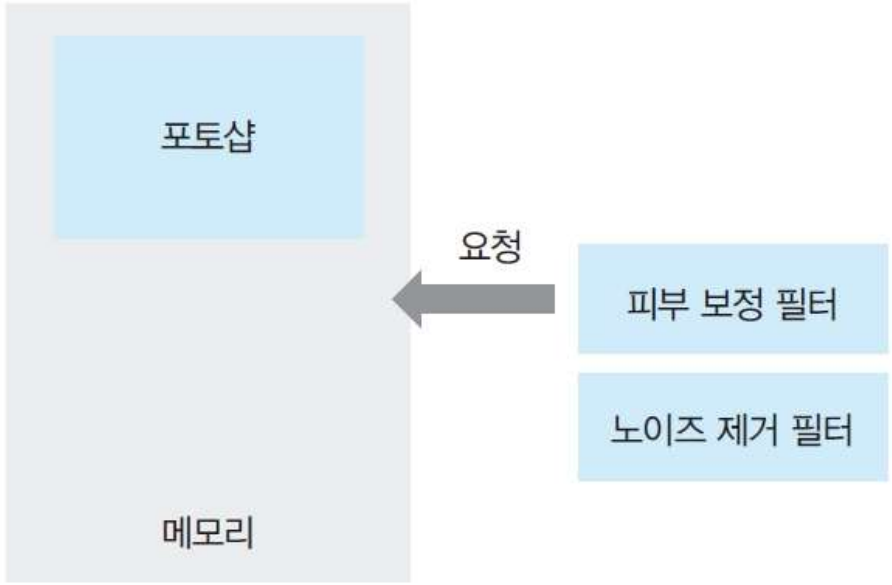


그림 9-1 요구 페이징의 필요성

1-1 요구 페이징의 개요

■ 요구 페이징(demand paging)

- 사용자가 요구할 때 해당 페이지를 메모리로 가져오는 것
- 미리 가져오기: 요구 페이징과 반대. 예상되는 페이지를 미리 가져 오는 방식으로 대표적인 예는 캐시. 미리 가져온 페이지를 쓰지 않으면 메모리 낭비
- 요구 페이징은 메모리의 절약과 효율적 관리, 프로세스의 응답 속도 향상 등의 효과를 볼 수 있음

1-2 페이지 테이블 엔트리의 구조

■ 요구 페이징과 스왑 영역

- 페이지가 스왑 영역에 있는 경우는 크게 두 가지
  - 요구 페이징으로 인해 처음부터 물리 메모리에 올라가지 못한 경우
  - 메모리가 꽉 차서 스왑 영역으로 옮겨 온 경우

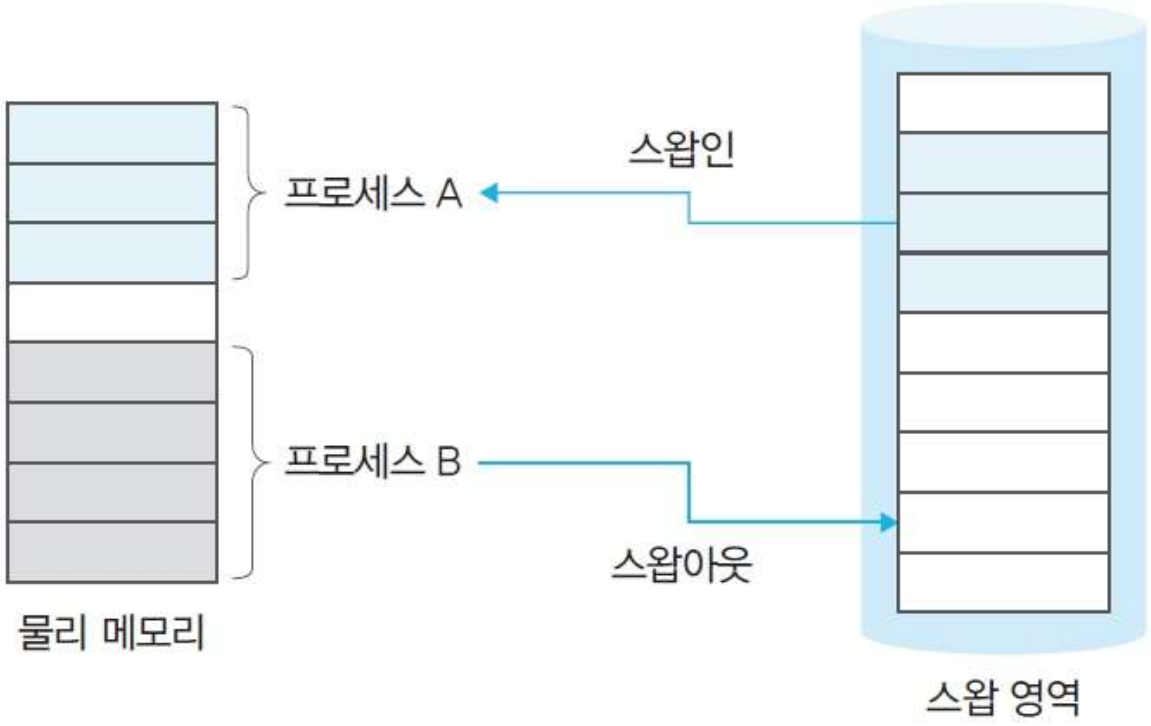


그림 9-2 스왑인과 스왑아웃

1-2 페이지 테이블 엔트리의 구조

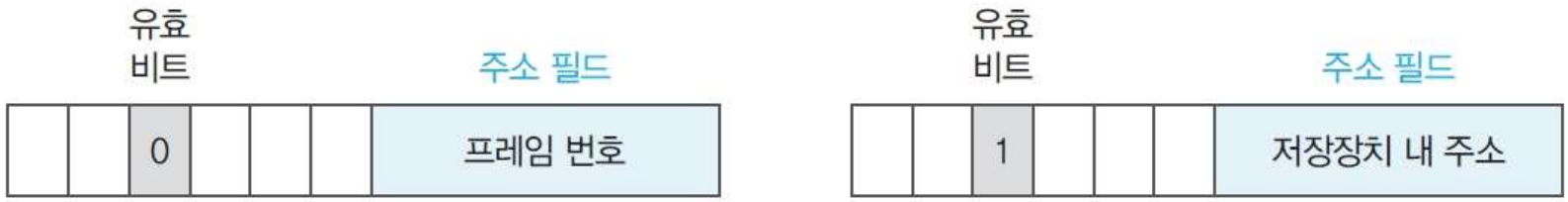
■ 페이지 테이블 엔트리(PTE)의 구성

그림 9-3 페이지 테이블 엔트리의 구성

- 페이지 번호
- 프레임 번호
- 플래그 비트
  - 접근 비트(access bit): 페이지가 메모리에 올라온 후 사용한 적이 있는지 알려주는 비트
  - 변경 비트(modified bit): 페이지가 메모리에 올라온 후 데이터의 변경이 있었는지 알려 주는 비트
  - 유효 비트(valid bit) : 페이지가 실제 메모리에 있는지를 나타내는 비트
  - 읽기(read bit), 쓰기(write bit), 실행 비트(execute bit): 페이지에 대한 읽기 권한, 쓰기 권한, 실행 권한을 나타내는 비트

1-3 페이지 부재

■ 유효 비트



(a) 유효 비트가 0일 때

(b) 유효 비트가 1일 때

그림 9-4 유효 비트에 따른 주소 필드의 변화

- 가상 메모리의 페이지 테이블에는 페이지가 물리 메모리에 있는지, 스왑 영역에 있는지 표시하기 위해 유효 비트를 사용
  - 유효 비트가 0: 페이지가 메모리에 있으므로 주소 필드에 물리 메모리의 프레임 번호 저장
  - 유효 비트가 1: 페이지가 스왑 영역에 있으므로 주소 필드에 스왑 영역 내 페이지 주소 저장

1-3 페이지 부재

■ 페이지 부재(page fault)

- 프로세스가 페이지를 요청했을 때 그 페이지가 메모리에 없는 상황
- 페이지 부재가 발생하면 프로세스가 해당 페이지를 사용할 수 있도록 스왑 영역에서 물리 메모리로 옮겨야 함

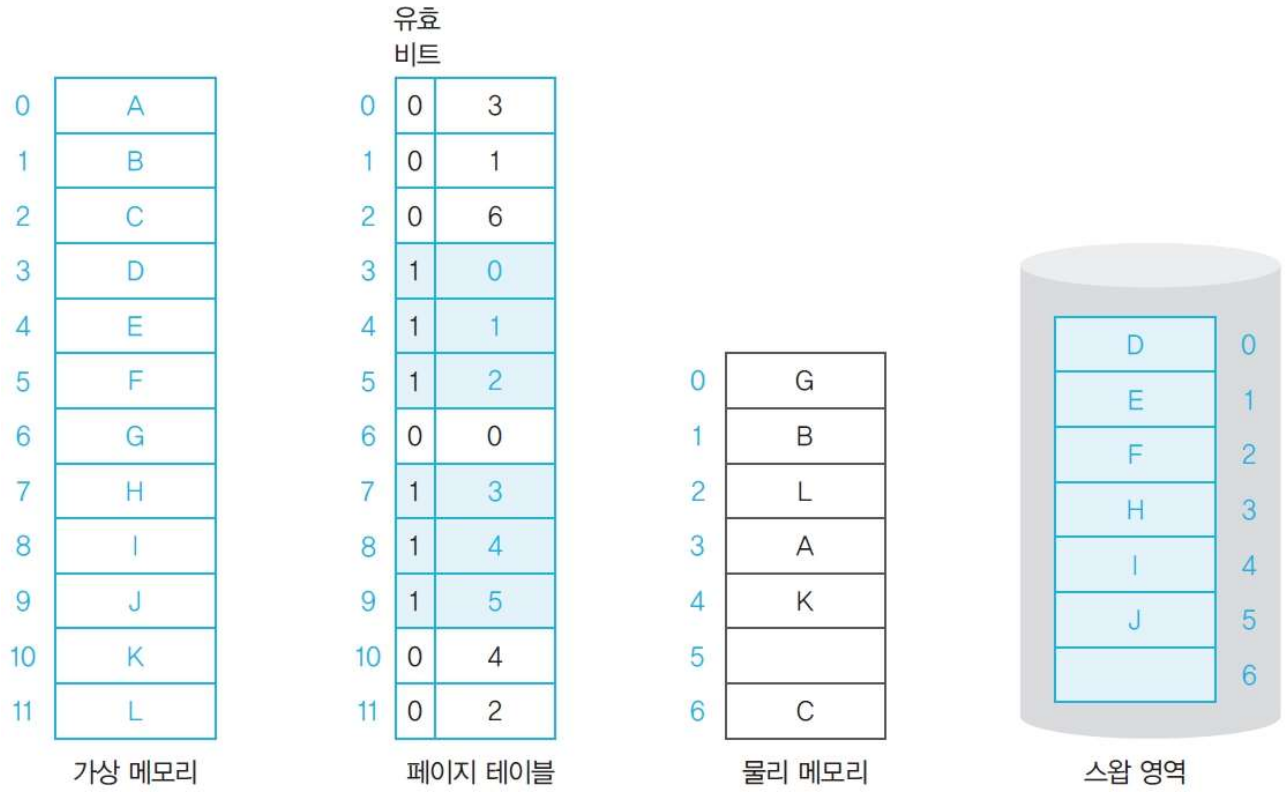


그림 9-5 물리 메모리와 스왑 영역에 저장된 페이지 매칭

1-3 페이지 부재

■ 페이지 부재 처리 과정

① 프로세스가 페이지 3 요청하면 페이지 테이블의 유효 비트가 1이므로 페이지 부재 발생

② 메모리 관리자는 스왑 영역의 0번에 있는 페이지를 메모리의 비어 있는 프레임인 5로 가져옴(스왑인)

③ 프레임 5로 접근하여 해당 데이터를 프로세스에 넘김

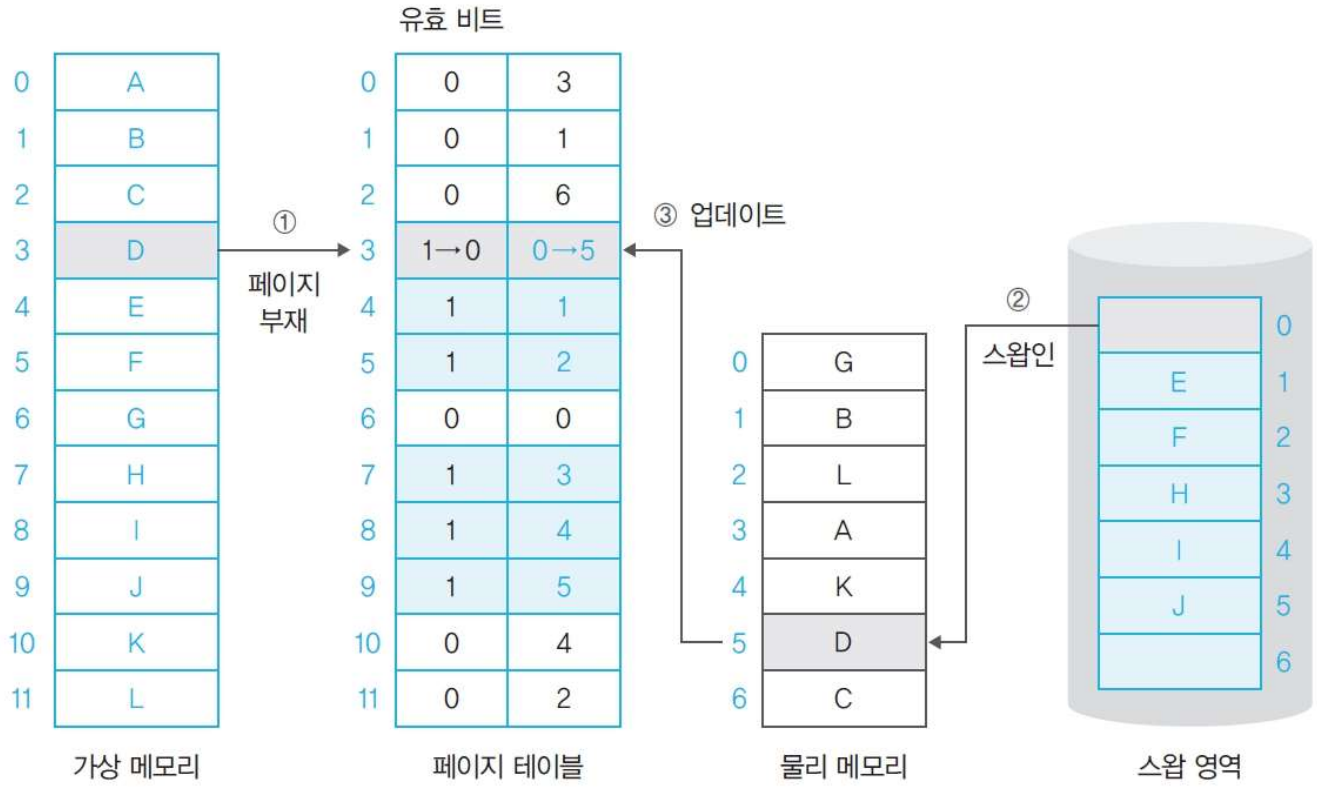


그림 9-6 페이지 부재 발생 시의 조치

1-3 페이지 부재

■ 페이지 교체

- 페이지 부재가 발생하면 스왑 영역에 있는 페이지를 메모리의 빈 영역에 올리고 페이지 테이블 갱신(업데이트)
- 빈 프레임이 없을 때는 메모리에 있는 프레임 중 하나를 스왑 영역으로 내보낸 후에 해당 페이지를 가져올 수 있음

■ 페이지 교체 알고리즘(page replacement algorithm)

- 어떤 페이지를 스왑 영역으로 내보낼지 결정하는 알고리즘

■ 대상 페이지(victim page)

- 페이지 교체 알고리즘에 의해 스왑 영역으로 보낼 페이지

1-3 페이지 부재

■ 메모리가 꽉 찬 상태에서 페이지 부재가 발생했을 때 조치

① 해당 페이지의 유효 비트가 1이므로 페이지 부재 발생

② 메모리가 꽉 차 있는 상태이기 때문에 스왑 영역에 있는 페이지 E를 가져오기 위해 메모리의 페이지 중 하나를 스왑 영역으로 내보내야 함(물리 메모리의 프레임 3에 저장된 페이지를 대상 페이지로 선정했는데 이 페이지를 스왑 영역으로 옮김(스왑아웃))

③ 대상 페이지 PTE 1의 유효 비트가 0에서 1로, 주소 필드 값이 프레임 3에서 스왑주소 6으로 바뀜

④ 스왑 영역 1번에 있던 페이지 E가 프레임 3으로 올라감(스왑인)

⑤ PTE 4의 유효 비트가 1에서 0으로, 주소 필드 값이 스왑 주소 1에서 프레임 3으로 바뀜

1-3 페이지 부재

■ 메모리가 꽉 찬 상태에서 페이지 부재가 발생했을 때 조치

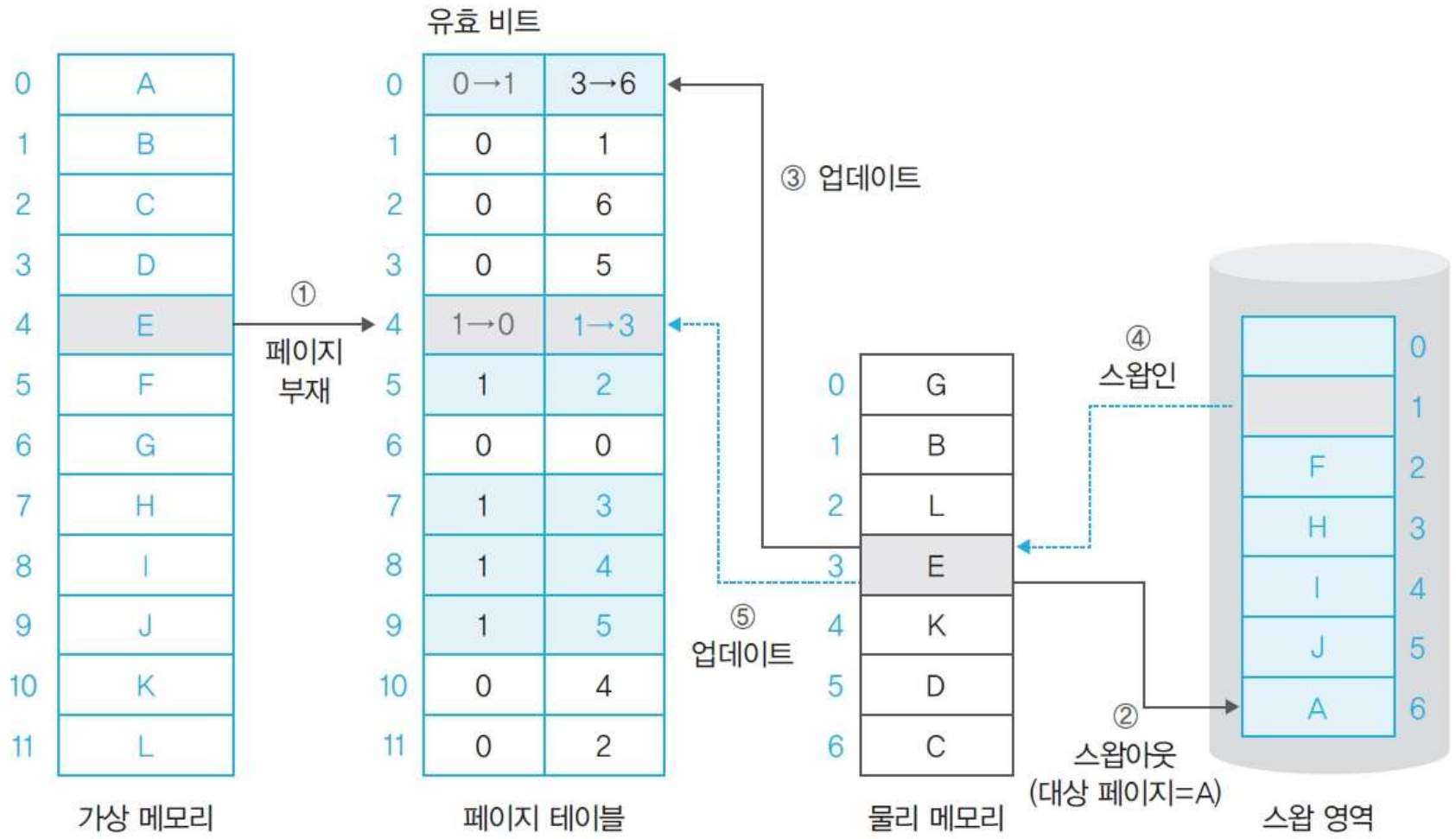


그림 9-7 대상 페이지의 스왑아웃

1-3 페이지 부재

■ 세그먼테이션 오류와 페이지 부재

- 세그먼테이션 오류(segmentation fault)

  사용자 프로세스가 주어진 메모리 공간을 벗어나거나 접근 권한이 없는 곳에 접근할 때 발생. 해당 프로세스를 강제 종료하여 해결

- 페이지 부재(page fault)

  해당 페이지가 물리 메모리에 없을 때 발생하는 오류. 사용자 프로세스와 무관하기 때문에 페이지 부재가 발생하면 메모리 관리자는 스왑 영역에서 해당 페이지를 물리 메모리로 옮긴 후 작업을 진행

2-1 페이지 교체 알고리즘의 개요

■ 페이지 교체 알고리즘의 종류

- 스왑 영역으로 보낼 페이지를 결정하는 알고리즘
- 메모리에서 앞으로 사용할 가능성이 적은 페이지를 대상 페이지로 선정하여 페이지 부재를 줄이고 시스템의 성능을 향상

표 9-1 페이지 교체 알고리즘의 종류

종류	알고리즘	특징
간단한 알고리즘	무작위	무작위로 대상 페이지를 선정하여 스왑 영역으로 보낸다.
	FIFO	처음 메모리에 올라온 페이지를 스왑 영역으로 보낸다.
이론적 알고리즘	최적	미래의 메모리 접근 패턴을 보고 대상 페이지를 선정하여 스왑 영역으로 보낸다.
최적 근접 알고리즘	LRU	시간적으로 멀리 떨어진 페이지를 스왑 영역으로 보낸다.
	LFU	사용 빈도가 적은 페이지를 스왑 영역으로 보낸다.
	NUR	최근에 사용한 적이 없는 페이지를 스왑 영역으로 보낸다.
	FIFO 변형	FIFO 알고리즘을 변형하여 성능을 높인다.

2-1 페이지 교체 알고리즘의 개요

■ 페이지 교체 알고리즘의 성능 평가 기준

- 어떤 알고리즘의 성능이 좋은지 비교 평가하는 방법은 다양함
- 같은 메모리 접근 패턴을 사용해 페이지 부재 횟수와 페이지 성공 횟수 비교함

요구	1	2	3	4	5	6	7	8	9	10
페이지	A	B	C	D	B	A	B	A	C	A

그림 9-8 공통으로 사용할 메모리 접근 패턴

2-2 무작위 페이지 교체 알고리즘

■ 무작위 페이지 교체 알고리즘(random page replacement algorithm)

- 스왑 영역으로 쫓아낼 대상 페이지를 특별한 로직 없이 무작위로 선정
- 지역성을 전혀 고려하지 않기 때문에 자주 사용하는 페이지가 대상 페이지로 선정되기도 하여 성능이 좋지 않음

2-3 FIFO 페이지 교체 알고리즘

■ FIFO 페이지 교체 알고리즘(First In First Out page replacement algorithm)

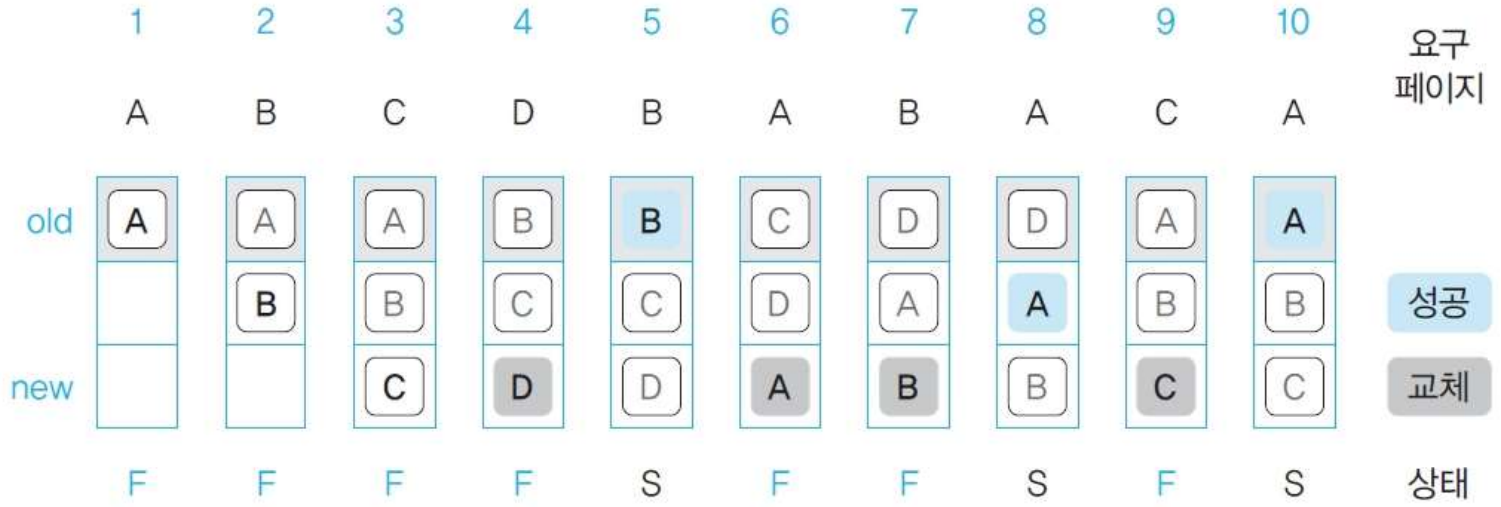


그림 9-9 FIFO 페이지 교체 알고리즘의 동작

- 시간상으로 메모리에 가장 먼저 들어온 페이지를 대상 페이지로 선정해 스왑 영역으로 쫓아냄
- 메모리가 꽉 차면 맨 위의 페이지가 스왑 영역으로 가고 나머지 페이지가 위쪽으로 이동, 새로운 페이지가 아래쪽 남은 공간에 들어옴
- 무조건 오래된 페이지를 대상 페이지로 선정하기 때문에 성능이 떨어짐

2-4 최적 페이지 교체 알고리즘

■ 최적 페이지 교체 알고리즘(optimal page replacement algorithm)

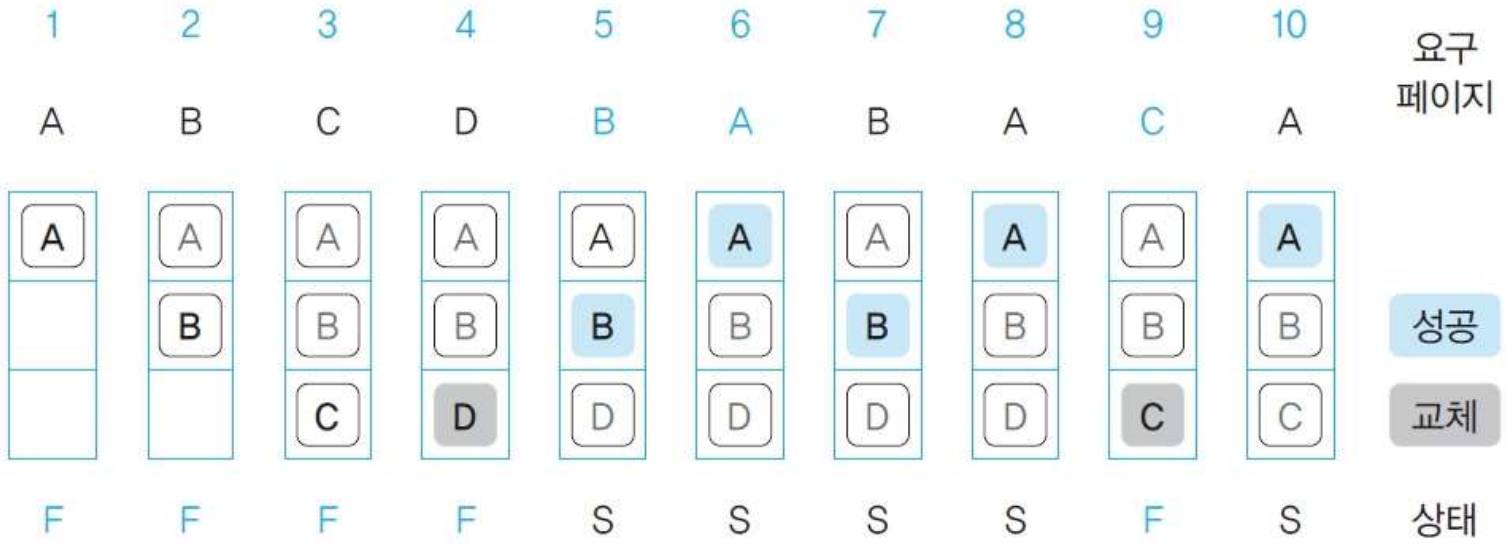


그림 9-10 최적 페이지 교체 알고리즘의 동작

- 앞으로 사용하지 않을 페이지를 스왑 영역으로 옮김
- 메모리가 앞으로 사용할 페이지를 미리 살펴보고 페이지 교체 선정 시점부터 사용 시점까지 가장 멀리 있는 페이지를 대상 페이지로 선정
- 이상적인 방법이지만 실제로 구현할 수 없음

2-4 최적 페이지 교체 알고리즘

■ 최적 근접 알고리즘(optimal approximation algorithm)의 방식

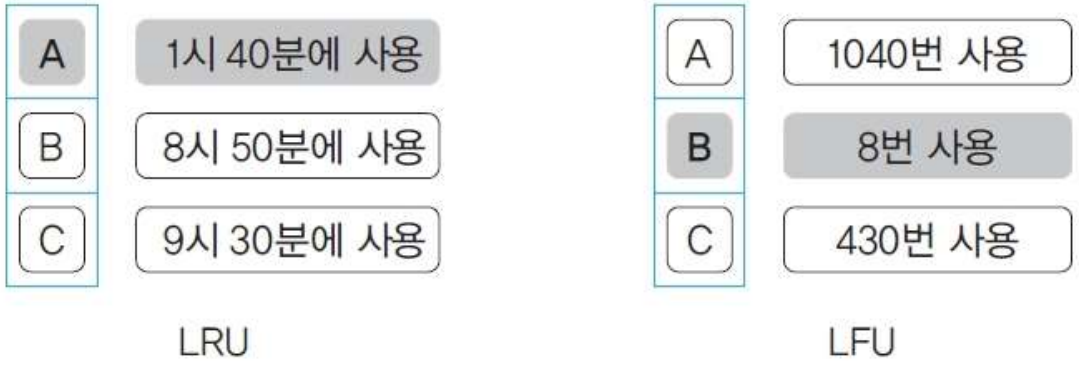


그림 9-11 최적 근접 알고리즘의 예

- 실제 구현이 가능하면서도 성능이 최적 근접 알고리즘에 근접한 알고리즘
- LRU(Least Recently Used) 페이지 교체 알고리즘: 페이지에 접근한 시간을 기준으로 대상 페이지 선정
- LFU(Least Frequently Used) 페이지 교체 알고리즘: 페이지가 사용된 횟수를 기준으로 대상 페이지 선정

2-5 LRU 페이지 교체 알고리즘

■ LRU 페이지 교체 알고리즘(Least Recently Used page replacement algorithm)

- '최근 최소 사용 페이지 교체 알고리즘'이라고도 함
- 메모리에 올라온 후 가장 오랫동안 사용되지 않은 페이지를 스왑 영역으로 옮김
- 최근 사용된 페이지는 놔두고 오래전 사용된 페이지를 대상 페이지로 선정
- 알고리즘은 시간을 기준으로 구현할 수 있으며 카운터나 참조 비트 이용하는 방법도 있음

2-5 LRU 페이지 교체 알고리즘

■ 페이지 접근 시간에 기반한 구현

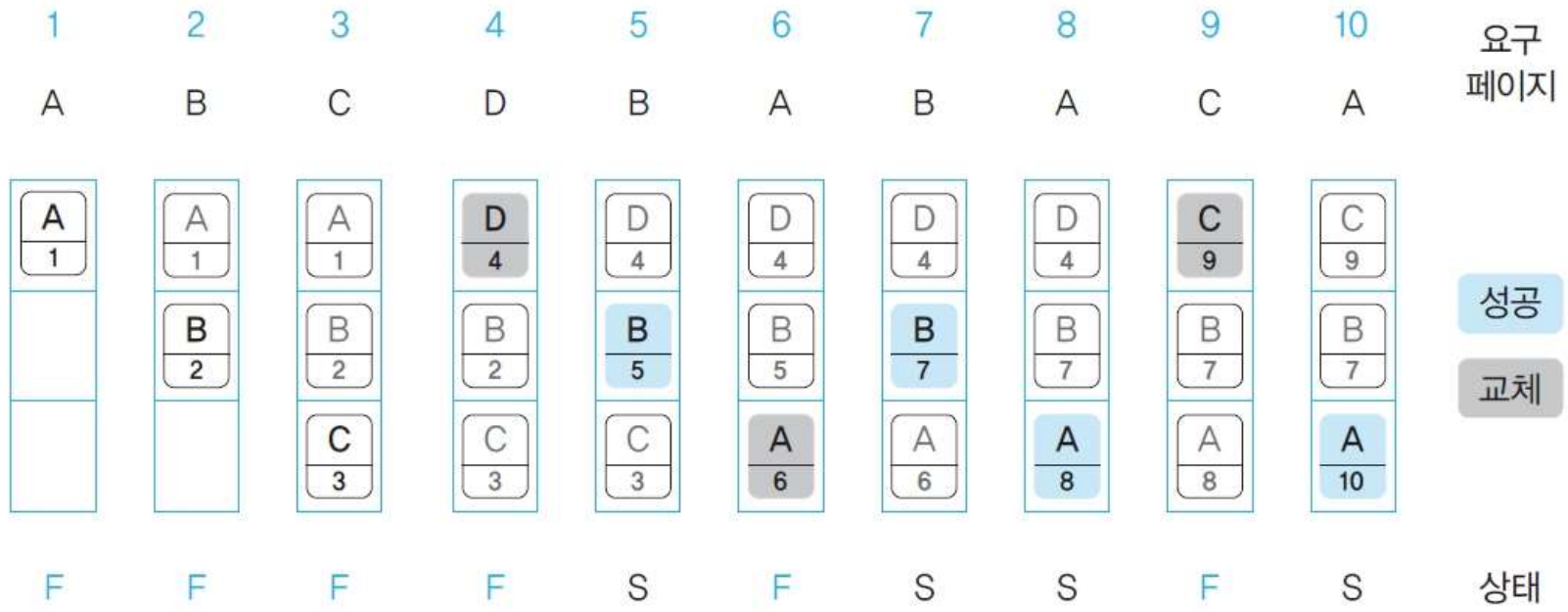


그림 9-12 LRU 페이지 교체 알고리즘의 동작

- 페이지에 접근한 지 가장 오래된 페이지를 교체

2-5 LRU 페이지 교체 알고리즘

■ 카운터에 기반한 구현

- 카운터를 사용하여 LRU 페이지 교체 알고리즘 구현

■ 참조 비트 시프트(reference bit shift) 방식

- 각 페이지에 일정 크기의 참조 비트를 만들어 사용하는 것
- 참조 비트의 초깃값은 0, 페이지에 접근할 때마다 1로 바뀜
- 참조 비트는 주기적으로 오른쪽으로 한 칸씩 이동(shift)
- 참조 비트를 갱신하다가 대상 페이지를 선정할 필요 있으면 참조 비트 중 **가장 작은 값**을 선정

그림 9-13 참조 비트 시프트 방식의 동작

2-5 LRU 페이지 교체 알고리즘

■ 참조 비트 시프트 방식이 LFU가 아니라 LRU 페이지 교체 알고리즘인 이유

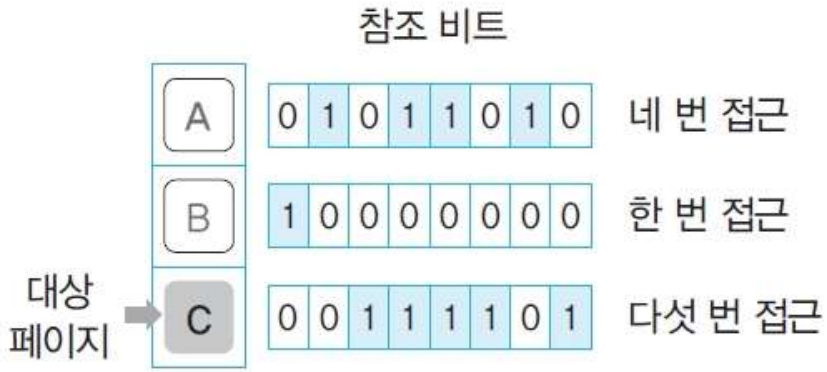


그림 9-14 참조 비트 시프트 방식의 대상 페이지 선정 기준

- 페이지 A는 4번, 페이지 B는 1번, 페이지 C는 5번 접근
- 가장 작은 값은 오랫동안 접근하지 않은 페이지 C가 대상 페이지
- 참조 비트 시프트 방식은 참조한 횟수 아닌 참조한 시간 기준으로 대상 페이지 선정하므로 LRU 페이지 교체 알고리즘의 한 방식으로 분류

2-6 LFU 페이지 교체 알고리즘

■ LFU 페이지 교체 알고리즘(Least Frequently Used page replacement algorithm)

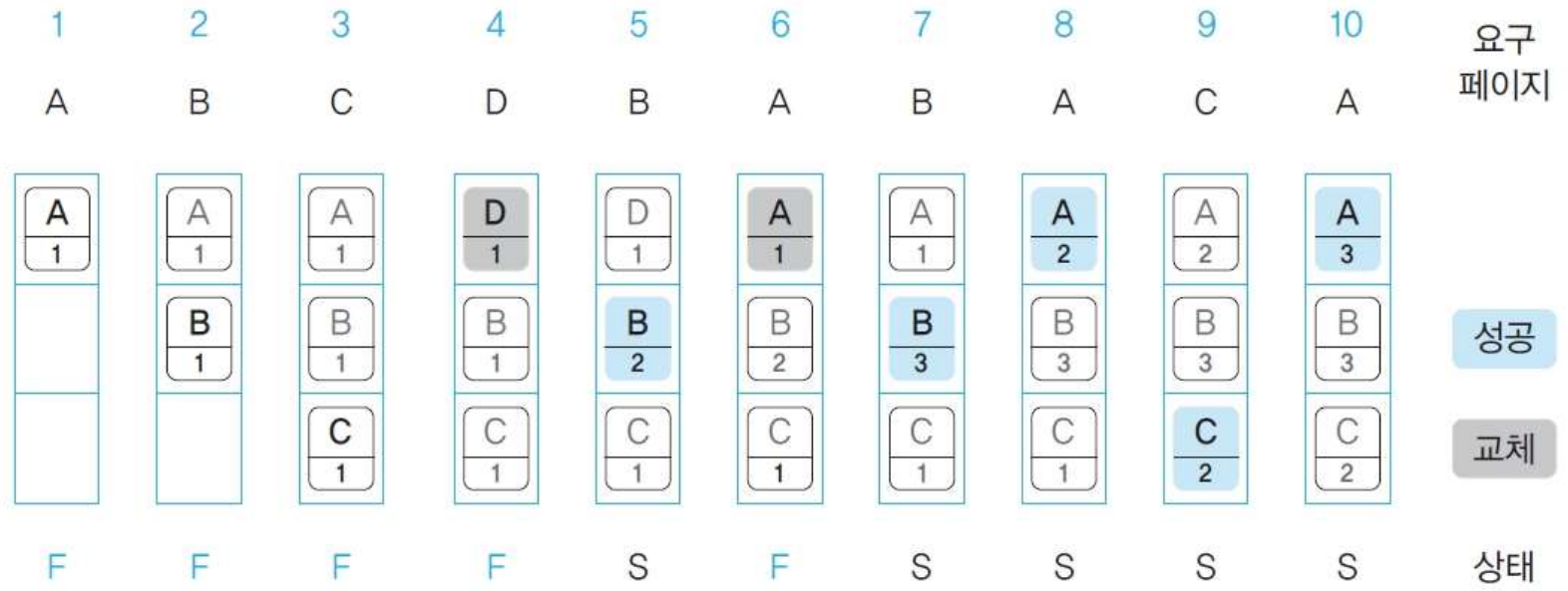


그림 9-15 LFU 페이지 교체 알고리즘의 동작

- 페이지가 몇 번 사용 되었는지를 기준으로 대상 페이지 선정
- 현재 프레임에 있는 페이지마다 그 동안 사용된 횟수를 세어 횟수가 가장 적은 페이지를 스왑 영역으로 옮김

2-7 NUR 페이지 교체 알고리즘

■ NUR 페이지 교체 알고리즘(Not Used Recently page replacement algorithm)

- LRU, LFU 페이지 교체 알고리즘은 성능 좋으나 추가 오버헤드가 큼!
- 이를 개선한 NRU는 두 개 비트만으로 구현 가능
- '최근 미사용 페이지 교체 알고리즘'이라고도 불림
- 페이지마다 참조 비트와 변경 비트를 가짐
  - 참조 비트: 페이지에 접근(read/execute)하면 1이 됨
  - 변경 비트: 페이지가 변경(write/append)되면 1이 됨

참조 비트	변경 비트	
0	0	가장 먼저 옮겨짐
0	1	
1	0	
1	1	모두 (1, 1)이면 초기화

그림 9-16 대상 페이지 선정 순서

2-7 NUR 페이지 교체 알고리즘

■ NUR 페이지 교체 알고리즘

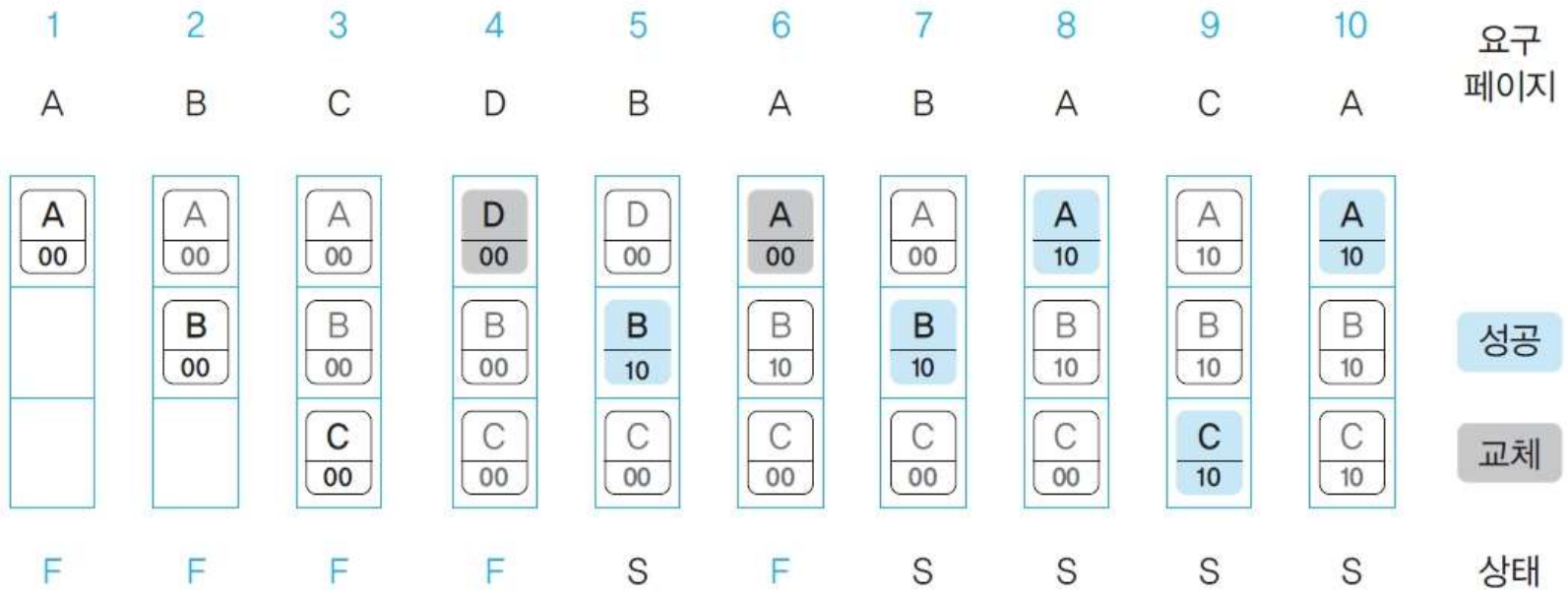


그림 9-17 NUR 페이지 교체 알고리즘의 동작

- 대상 페이지 찾을 때 참조 비트가 0인 페이지를 먼저 찾고, 없으면 변경 비트가 0인 페이지를 찾음
- 같은 비트의 페이지가 여러 개라면 무작위로 대상 페이지 선정

2-8 FIFO 변형 알고리즘

■ 2차 기회 페이지 교체 알고리즘(second chance page replacement algorithm)

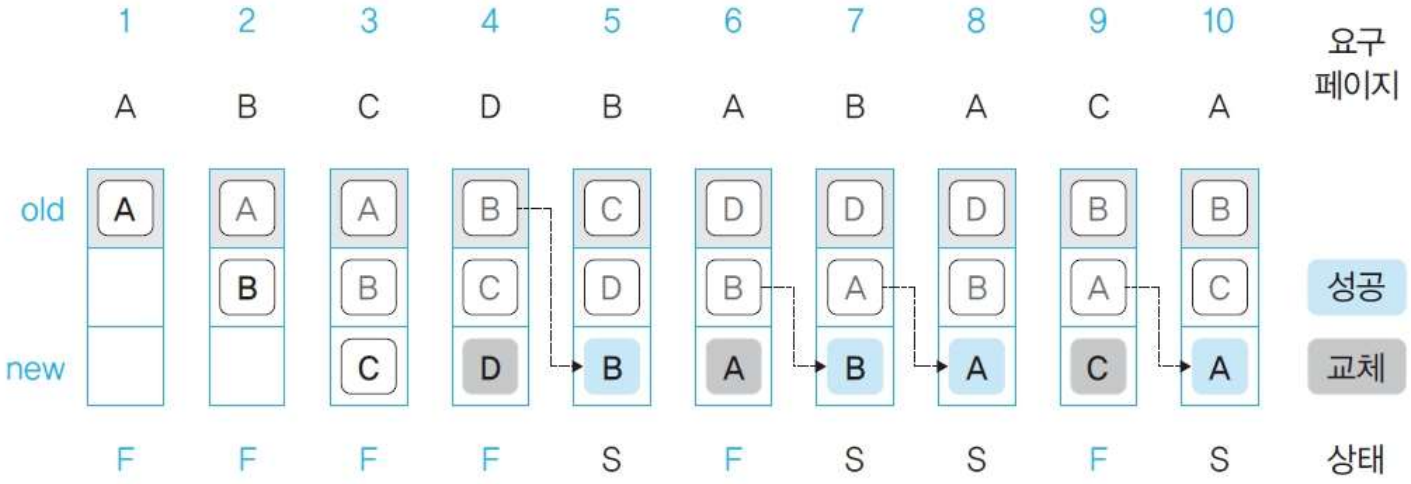


그림 9-18 2차 기회 페이지 교체 알고리즘의 동작

- FIFO 페이지 교체 알고리즘과 마찬가지로 큐 사용
- 특정 페이지에 접근하여 페이지 부재 없이 성공할 경우 해당 페이지를 큐의 맨 뒤로 이동하여 대상 페이지에서 제외
- 성공한 페이지를 큐의 맨 뒤로 옮김으로써 기회를 한 번 더 줌

2-8 FIFO 변형 알고리즘

■ 시계 알고리즘

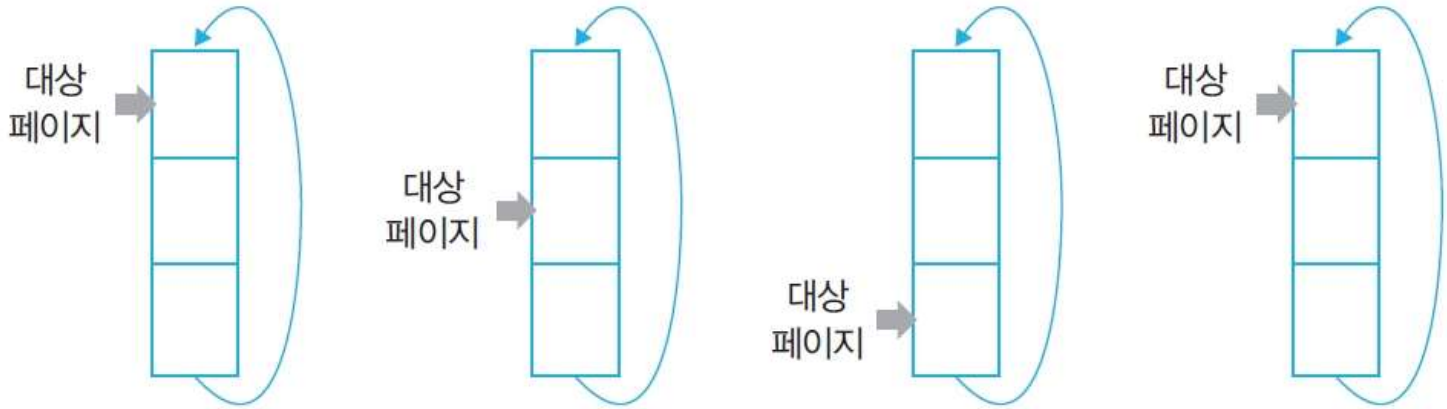


그림 9-19 시계 알고리즘의 특징

- 2차 기회 페이지 교체 알고리즘은 큐 사용, 시계 알고리즘은 원형 큐 사용
- 스왑 영역으로 옮길 대상 페이지를 가리키는 포인터 사용
- 포인터가 큐의 맨 아래로 내려가면 다음번에는 다시 큐의 처음을 가리킴

2-8 FIFO 변형 알고리즘

■ 시계 알고리즘의 동작

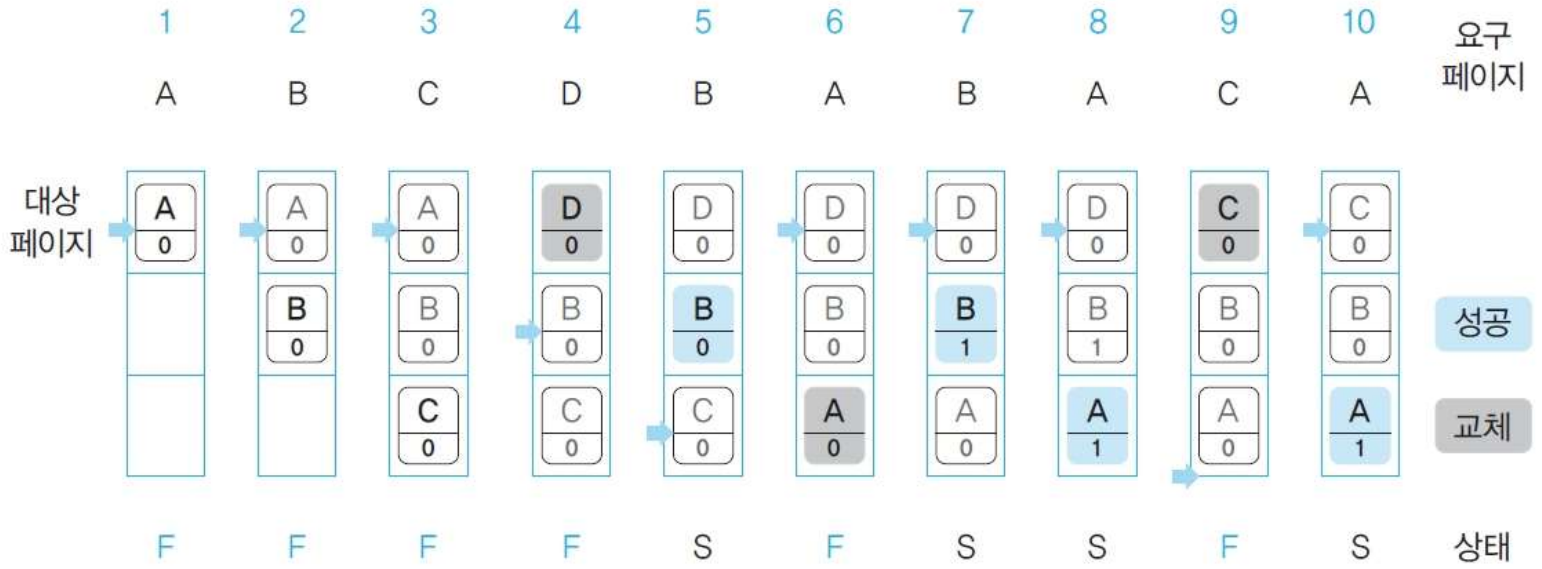


그림 9-20 시계 알고리즘의 동작

- 각 페이지에 참조 비트가 하나씩 추가(2차 기회 페이지 교체 알고리즘과 비교했을 때)
- 참조 비트의 초깃값은 0, 메모리에 있는 페이지를 성공적으로 참조하면 0에서 1로 변경
- 대상 포인터는 메모리가 꽉 찰 경우 스왑 영역으로 쫓겨날 페이지를 가리킴
- 가리키는 페이지가 스왑 영역으로 쫓겨 나면 대상 포인터를 밑으로 이동, 이때 참조 비트가 1인 페이지는 0으로 만든 후 건너 뜀(2차 기회를 줌)

3-1 스레싱

■ 스레싱(threshing)의 개념

- 하드디스크 입출력이 너무 많아져 잦은 페이지 부재로 작업이 멈춘 것 같은 상태

■ 물리 메모리의 크기와 스레싱

- 스레싱 발생 지점(threshing point): CPU 작업 시간보다 스왑 영역으로 페이지 보내고 새로운 페이지를 메모리에 가져오는 작업이 빈번해져 CPU가 작업할 수 없는 상태에 이르게 되는 시점
- 물리 메모리 크기 늘리면 스레싱 발생 지점이 늦춰져 프로세스를 원만하게 실행

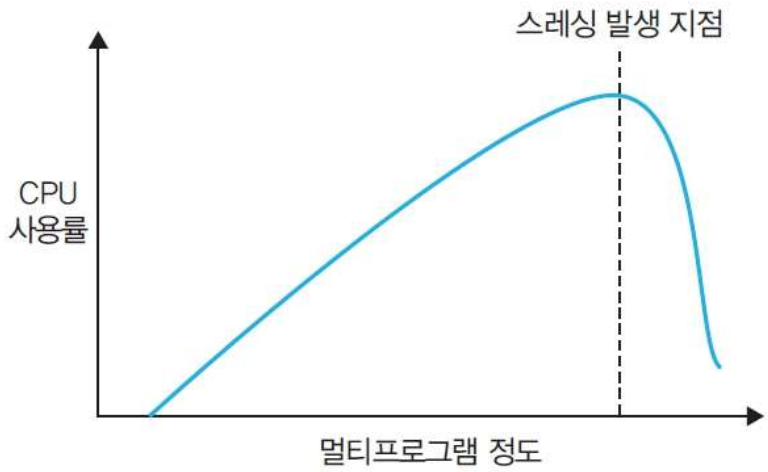


그림 9-22 멀티프로그래밍 정도와 스레싱

3-1 스레싱

■ 스레싱과 프레임 할당

- 프로세스에 너무 적은 프레임을 할당하면 페이지 부재가 빈번히 일어남
- 프로세스에 너무 많은 프레임을 할당하면 페이지 부재는 줄지만 메모리 낭비
- 프로세스에 프레임을 할당하는 방식은 정적 할당과 동적 할당으로 구분

3-2 정적 할당

■ 균등 할당(equal allocation)

- 프로세스 크기와 상관없이 사용 가능한 프레임을 모든 프로세스에 동일하게 할당
- 크기가 큰 프로세스는 필요한 만큼 프레임을 할당받지 못해 페이지 부재가 빈번하게 발생, 크기가 작은 프로세스는 메모리 낭비

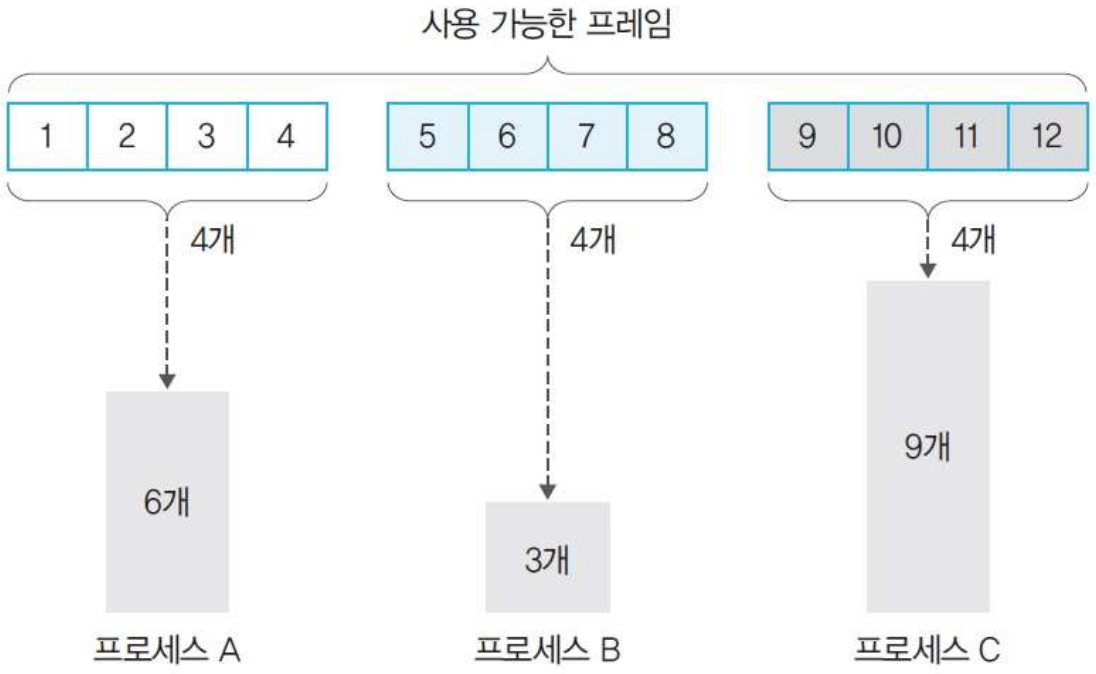


그림 9-23 균등 할당 방식

3-2 정적 할당

■ 비례 할당(proportional allocation)

- 프로세스 크기에 비례하여 프레임 할당
- 프로세스 크기를 고려하지 않는 고정 할당보다 좀 더 현실적인 방식
  - 프로세스 실행 중에 필요한 프레임을 유동적으로 반영하지 못함
  - 사용하지 않을 메모리를 처음부터 미리 확보하여 공간 낭비

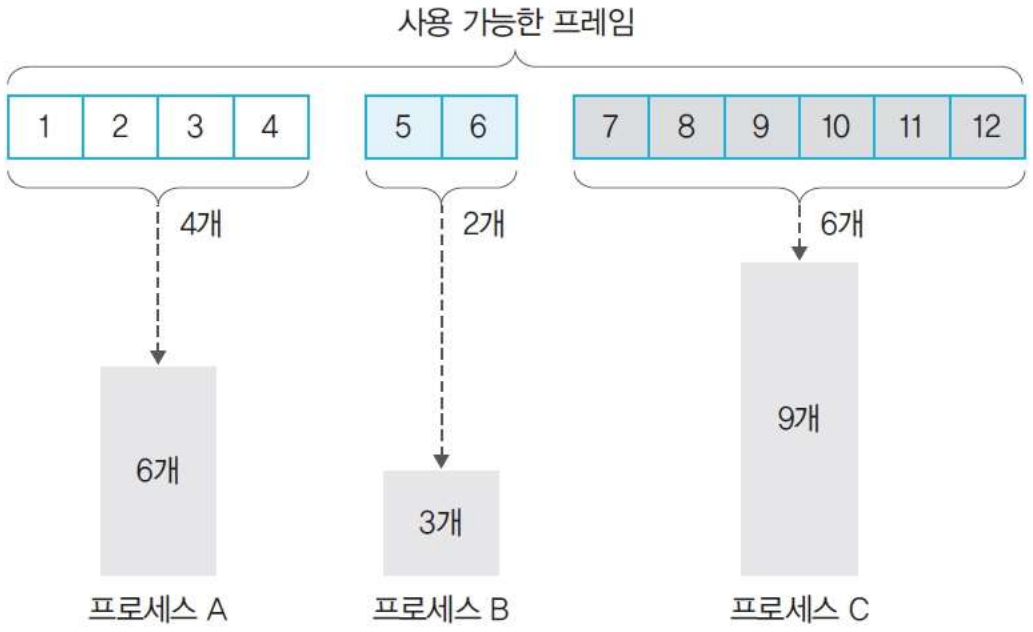


그림 9-24 비례 할당 방식

3-3 동적 할당

■ 작업집합 모델(working set model)

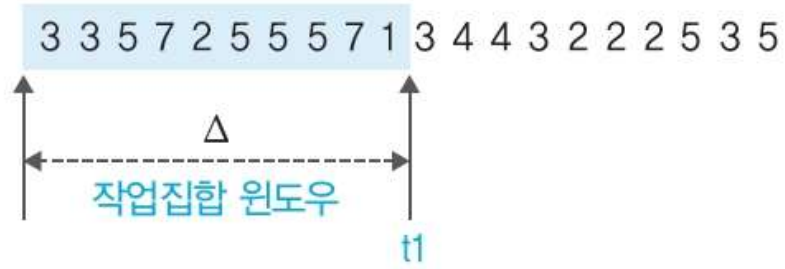


그림 9-25 작업집합 모델

- 최근 일정 시간 동안 참조된 페이지를 집합으로 만들고, 이 집합에 있는 페이지를 물리 메모리에 유지
  - 작업집합 크기(working set size): 작업집합 모델에서 물리 메모리에 유지할 페이지 크기
  - 작업집합 윈도우(wsw; Working Set Window): 작업집합에 포함되는 페이지 범위
- 델타 동안 참조된 10개 페이지 중 작업집합에는 WS(t1)={1, 7, 5, 2, 3} 삽입되고, 이 페이지들은 다음 윈도우에 도달할 때까지 물리 메모리에 보존

3-3 동적 할당

■ 작업집합의 갱신

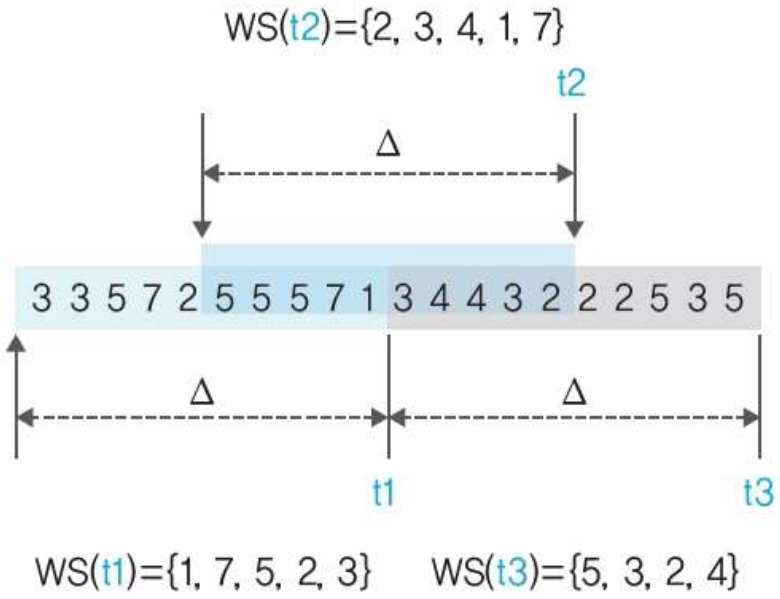


그림 9-26 작업집합 모델에서 시간에 따른 집합의 변화

- 작업집합 크기가 5라는 것은 페이지에 다섯 번 접근할 때마다 작업 집합을 갱신한다는 의미

3-3 동적 할당

■ 작업집합 윈도우의 크기와 프로세스 실행 성능

- 작업집합 윈도우를 너무 크게 잡으면 필요 없는 페이지가 메모리에 남아 다른 프로세스에 영향을 미침
- 너무 작게 잡으면 필요한 페이지가 스왑 영역으로 옮겨져 프로세스 성능이 떨어짐
- 적정크기의 작업집합을 유지하여 메모리를 효율적으로 관리할 수 있음

3-3 동적 할당

■ 페이지 부재 빈도(page fault frequency)

- 페이지 부재 횟수를 기록하여 페이지 부재 비율을 계산하는 방식
- 페이지 부재 비율이 상한선을 초과하면 프레임을 추가하여 늘림
- 페이지 부재 비율이 하한선 밑으로 내려가면 할당한 프레임을 회수
- 페이지 부재 빈도 방식은 프로세스를 실행하면서 추가로 페이지를 할당하거나 회수하여 적정 페이지 할당량을 조절

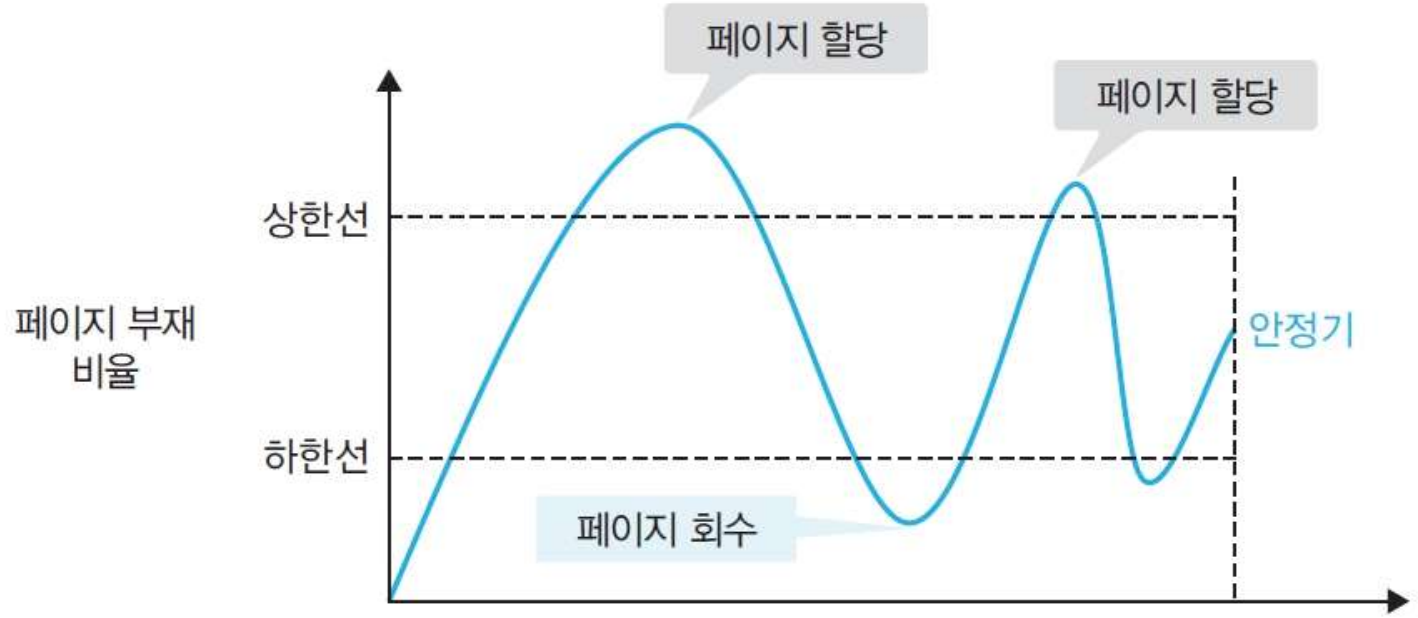


그림 9-27 페이지 부재 비율과 페이지 할당

3-4 전역 교체와 지역 교체

■ 전역 교체와 지역 교체

그림 9-28 전역 교체와 지역 교체의 차이

- 전역 교체: 전체 프레임을 대상으로 교체 알고리즘 적용
- 지역 교체: 현재 실행 중인 프로세스의 프레임을 대상으로 교체 알고리즘 적용

3-4 전역 교체와 지역 교체

■ 지역 교체 방식의 장단점

- 장점: 자신에게 할당된 프레임의 전체 개수에 변화가 없기 때문에 페이지 교체가 다른 프로세스에 영향을 미치지 않음
- 단점: 자주 사용하는 페이지가 스왑 영역으로 옮겨져 시스템 효율이 떨어짐