가상 메모리 관리

12th Week

Kim, Eui-Jik

Contents

- 소개
- 지역성
- 요구 페이징
- 예측 페이징
- 페이징 교체
- 페이징 교체 전략

소개

- 소개
  - 가상 메모리 페치 전략
    - 페이지나 세그먼트를 2차 저장소에서 메인 메모리로 가져올 시점 결정
  - 요구 페치 전략
    - 프로세스가 페이지나 세그먼트를 참조할 때까지 대기했다가 메인 메모리에 로드
  - 예측 페치 전략
    - 프로세스가 잠시 후 어떤 페이지 혹은 세그먼트를 참조할지 미리 예측
    - 참조 가능성 높고, 메모리 공간이 여유 있을 시, 시스템은 해당 페이지 혹은 세그먼트를 프로세스가 명시적으로 참조하기 전에 메인 메모리에 로드
      - 참조가 일어날 경우 프로세스가 대기할 필요 없어 성능 개선
  - 교체 전략
    - 유입되는 페이지나 세그먼트를 위한 공간을 내주기 위해 어떤 페이지, 어떤 세그먼트를 교체할지 결정

지역성

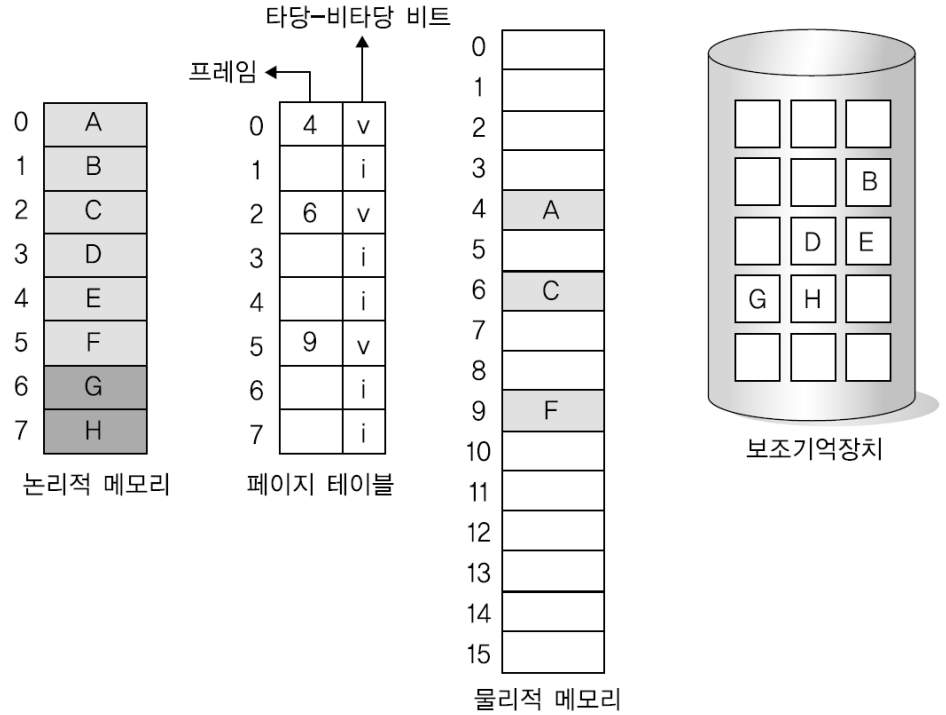
- 지역성
  - 프로세스가 편중된 패턴으로 메모리를 참조하는 경향
  - 시간적 지역성
    - 시간에 편중되는 현상
    - 참조된 기억장소는 가까운 미래에도 계속 참조될 가능성이 높음을 의미함
  - 공간적 지역성
    - 근처에 있는 항목이 비슷한 경향이 있음을 의미
    - 프로세스가 어떤 기억장소를 한 번 참조하면, 이후에 참조한 기억장치 근처에 있는 기억장소를 참조할 가능성이 높음을 의미함

요구 페이징

- 요구 페이징
  - 가상 메모리 시스템에서 구현하는 가장 간단한 페치 정책
  - 프로세스가 처음 실행될 때 해당 명령어를 담은 페이지를 메인 메모리로 로드
    - 프로세스가 명시적으로 페이지를 참조할 때 해당 페이지들을 2차 저장소에서 메인 메모리로 로드
  - 페이지를 한번에 하나씩 축적
    - 새로운 페이지 참조 시, 프로세스는 시스템이 해당 페이지를 메인 메모리로 가져올 때 까지 대기

요구 페이징

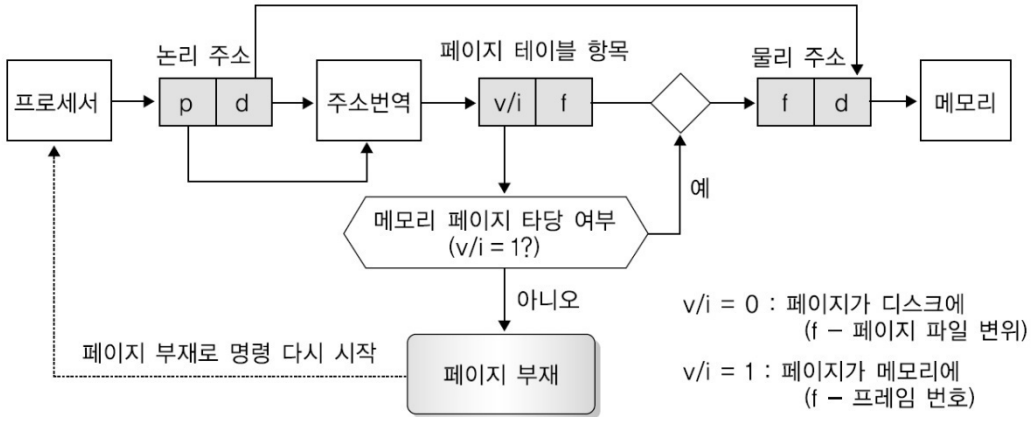
- 요구 페이징 기본 개념
  - 타당-비타당을 나타내는 bit를 페이지 테이블의 각 항목에 추가함
    - 타당/비타당 비트가 추가된 페이지 테이블
      - 타당 비트(v)로 설정된 경우 해당 페이지가 물리적 메모리에 있음을 의미함
      - 비타당 비트(i)로 설정된 경우 페이지는 보조기억장치(디스크)에 있음을 의미함
      - 저장되지 않은 페이지 사용 시 페이지 부재(Page Fault) 발생하며, 이때 액세스 방지를 위해 타당/비타당 비트를 이용하여 처리함



타당/비타당 비트가 추가된 페이지 테이블

요구 페이징

- 비타당 비트
  - 프로그램이 비타당 비트로 표시된 페이지에 액세스 하지 않는 경우 실행에 영향을 주지 않음
  - 프로그램이 액세스하는 경우 페이지 부재로 운영체제에 트랩 발생
    - 무효 주소에 의한 오류
- 페이지 부재 (Page Fault)
  - [그림] 페이징 시스템의 페이지 부재
    - 프로세서에 의해 생성되는 가상(논리) 주소 V=(p, d)에서 페이지 번호(p)는 주소번역(페이지 테이블 레지스터)과정을 통해 페이지 테이블에서 해당 페이지 테이블 항목에 접속
    - 메모리에 지정된 프레임의 저장 여부 확인
    - 지정된 프레임 페이지가 저장되어 있는 경우 프레임 번호와 변위 결합, 물리주소 생성
    - 그렇지 않은 경우, 페이지 부재로 해당 페이지(프레임)가 디스크에 저장되어 있음을 의미하므로 명령 재시작

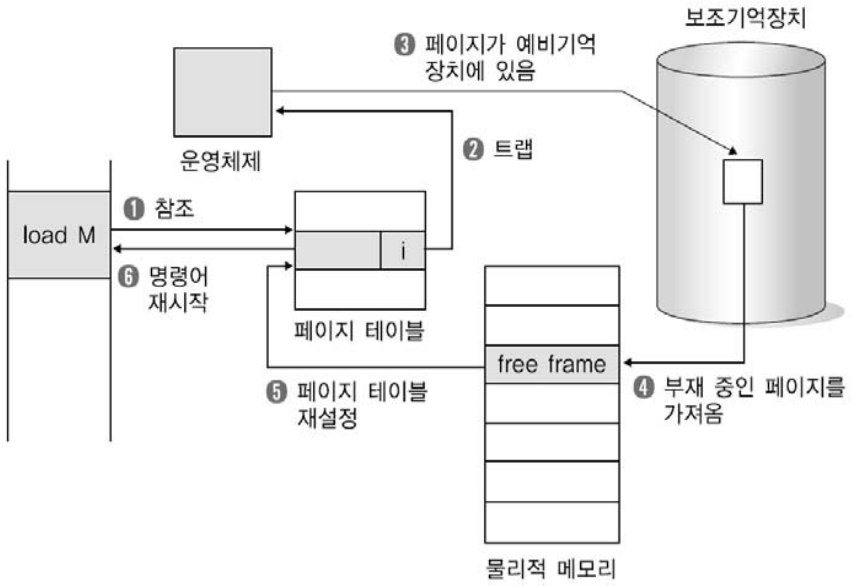


페이징 시스템의 페이지 부재

[참고]

요구 페이징

- 페이지 부재 처리 과정
  - ① 프로세스 제어 블록에 있는 내부 테이블 검사, 프로세스 참조가 메모리 액세스에 타당한지 부당한지 결정함
  - ② 프로세스 참조가 무효화될 경우 프로세스는 중단. 프로세스가 유효한 참조이나 페이지를 가져오지 않은 경우 페이지를 가져옴
  - ③ 비어있는 프레임 리스트 중 하나를 선택
  - ④ 할당된 프레임에 요구된 페이지를 읽어 들이기 위해 디스크 동작을 스케줄함
  - ⑤ 요구된 페이지가 메모리에 있다는 것을 알리기 위해 페이지 테이블을 수정
  - ⑥ 주요 트랩에 의해 인터럽트된 명령어들을 다시 시작



페이지 부재 처리과정

요구 페이징

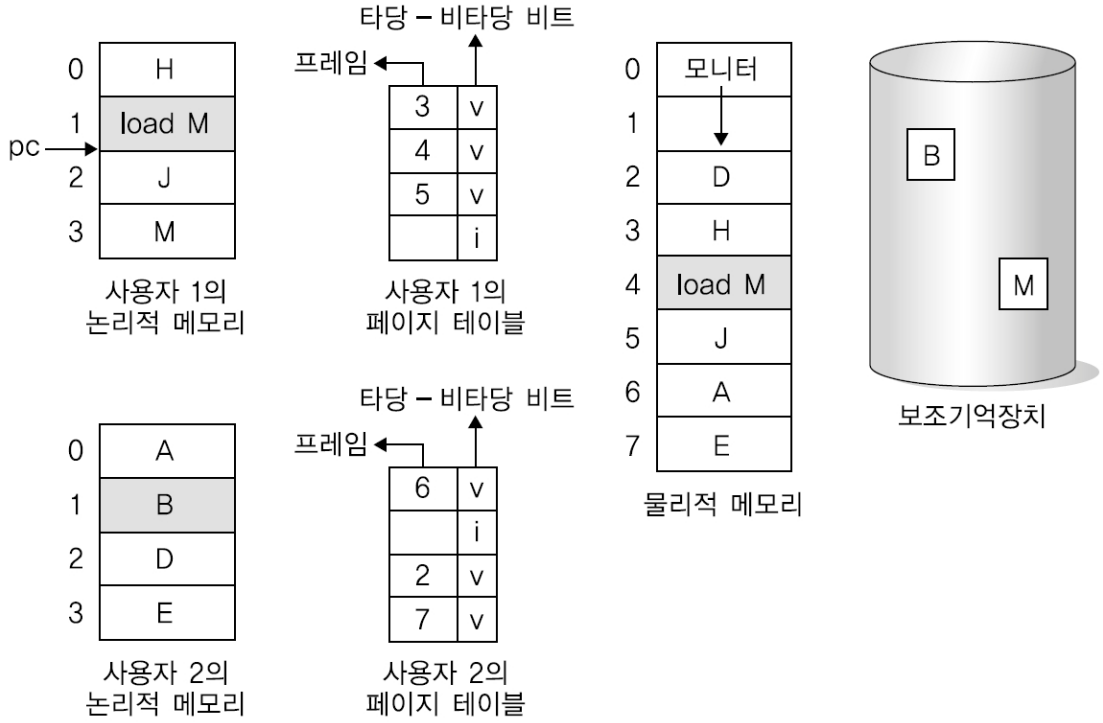
- 장단점
  - 장점
    - 다중 프로그래밍의 정도를 증가시키고 액세스되지 않은 페이지를 로드하지 않아 메모리가 절약됨
    - 프로그램을 시작할 때 적재(로딩) 지연이 적음
    - 적은 수의 페이지를 읽으므로 초기 디스크 오버헤드가 적음
    - 적재된 페이지들 중 하나가 수정될 때까지 페이지들은 여러 프로그램에 의해 공유되므로 공유 페이지 (코드 공유) 기술은 보다 많은 자원 절약 가능
    - 프로그램을 실행할 충분한 메모리가 없는 시스템에서도 대용량 프로그램을 실행 가능하며, 프로그래머는 이전 중첩(오버레이) 기법보다 쉽게 구현 가능함
  - 단점
    - 개별 프로그램들은 페이지에 처음 액세스할 때 약간의 지연이 발생함
    - 낮은 비용, 낮은 성능의 시스템에 실행되는 프로그램은 페이지 대체를 지원하는 메모리 관리 장치가 없음
    - 메모리 관리(페이지 교체)가 복잡함

요구 페이징

- 페이지 대치
  - 페이지 대치의 필요성
    - [그림] 페이지 대치의 필요성

      - 프로그램 카운터(PC)에 의해 사용자 1의 페이지 M을 적재하면 물리적 메모리에 비어있는 프레임이 없음을 발견

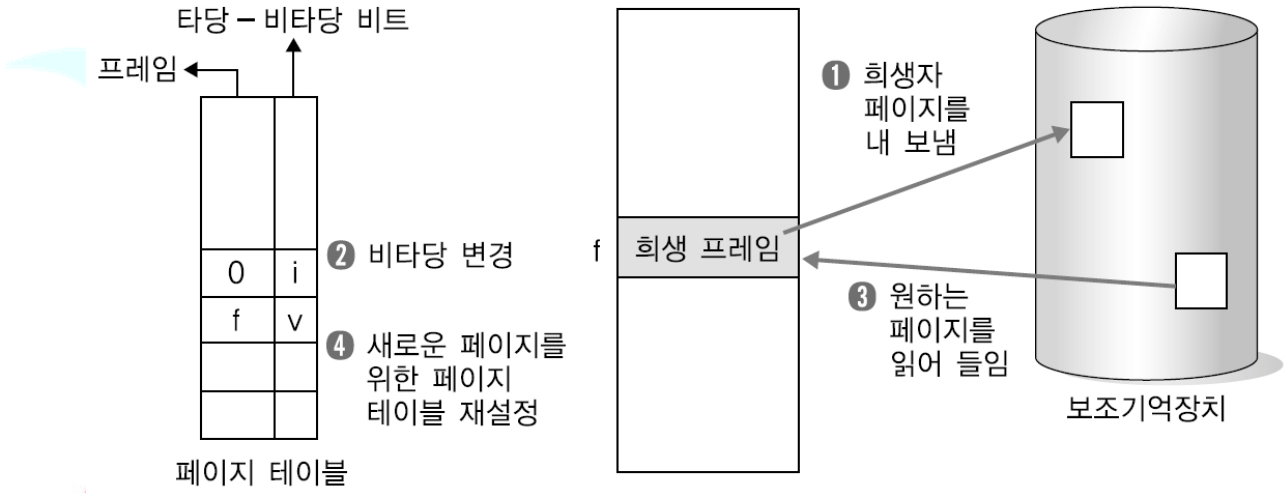
      - 사용자 2의 페이지 B는 물리적 메모리에 적재할 수 없음



페이지 대치의 필요성

요구 페이징

- 페이지 대치 기법
  - 페이지 방식을 취하는 가상 메모리에서 페이지 부재 발생 시 메인 메모리에 있으면서 사용되지 않는 페이지를 없애고 새로운 페이지로 변경
  - 아래와 같은 방식으로 액세스 됨
  - ① 비어 있는 프레임이 없다면 현재 사용하지 않는 프레임(희생자)을 찾음. 사용하지 않는 프레임이 있는 경우 발견한 프레임을 비우기 위해 프레임의 내용을 보조기억장치(디스크)에 저장
  - ② 페이지가 메모리에 더 이상 존재하지 않는다는 것을 알려주기 위해 페이지 테이블을 변화(비타당 비트로 전환)시킴으로써 프레임이 비게 됨
  - ③ 원하는 페이지(f)를 디스크로부터 읽어 프레임에 저장
  - ④ 새로운 페이지(f)를 위하여 페이지 테이블을 수정함



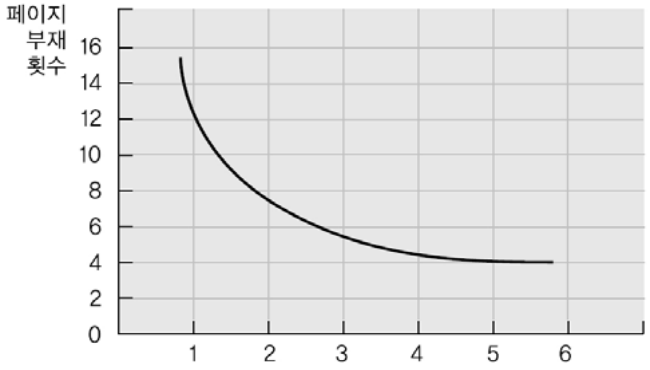
페이지 대치

페이지 대치 알고리즘

- 페이지 부재와 프레임 개수
  - 참조 문자열을 이용한 페이지 부재 횟수와 프레임 개수와의 관계
    - 참조문자열은 메모리를 참조하는 페이지를 의미함

참조문자열:1, 4, 1, 6, 1,6,1,6,1,6,1

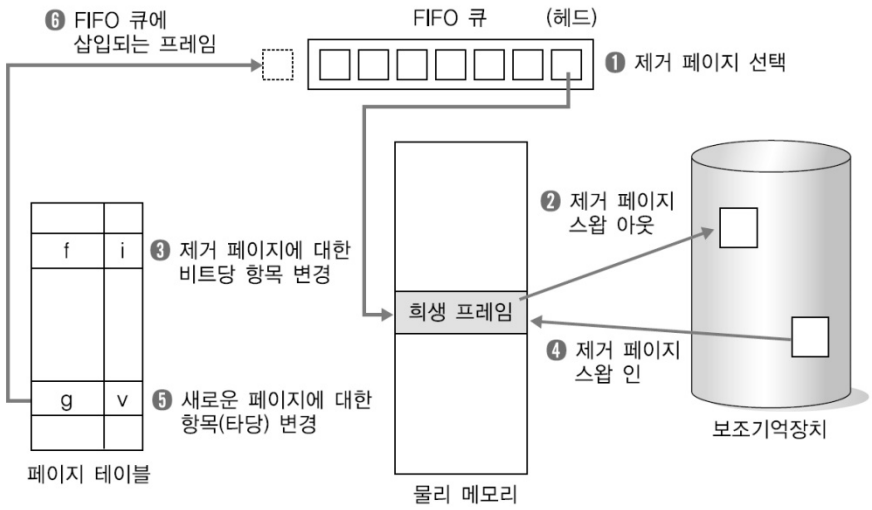
    - 프레임 수가 증가하면 페이지 부재수가 감소함
      - 위의 참조 문자열에 대해 세 개 또는 그 이상의 프레임을 가진 경우, 총 3번의 페이지 부재 발생
      - 한 프레임만 이용할 수 있는 경우 각 참조마다 페이지 대치가 필요하며 결과적으로 11번의 부재 발생
    - 페이지 부재와 프레임 개수 그래프
      - 프레임 수가 증가함에 따라 페이지 부재의 수가 최소한도의 수준으로 내려감



페이지 부재와 프레임 개수

페이지 대치 알고리즘 - FIFO

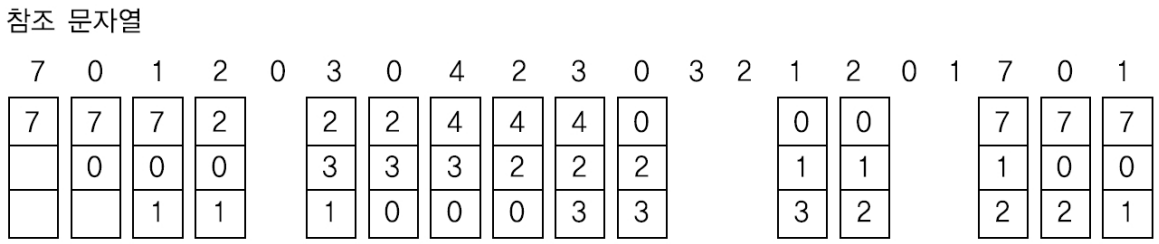
- 선입선출(FIFO, First-In-First-Out) 대치 알고리즘
  - 각 페이지가 메모리 안으로 들어간 시간을 이용
    - 가장 오래된 페이지부터 우선 대치시킴
    - 페이지 부재 발생 시, 즉 제거해야 할 페이지 선택 후 보조기억장치로 이동 교체시키고 페이지 테이블의 타당/비타당 비트를 변경함
    - 새로운 페이지에 대한 페이지 테이블 항목 변경 후 FIFO 큐의 마지막 위치에 삽입



선입선출(FIFO) 대치 알고리즘

페이지 대치 알고리즘 - FIFO

- 선입선출 큐(FIFO Queue)에 의해 메모리의 모든 페이지가 관리됨
  - 큐의 헤드부분에 있는 페이지를 먼저 대치함
  - 큐에 있는 페이지가 메모리로 들어갈 때 큐의 끝 페이지가 삽입
  - 큐의 크기는 사용 가능한 메모리 프레임의 수에 해당됨
  - 선입선출(FIFO) 대치 알고리즘의 실행
    - 세 개의 프레임을 사용할 수 있다 가정하고 참조 문자열을 실행, 페이지 부재는 15번 발생함
- 알고리즘의 이해가 쉽고 프로그램의 작성이 쉬움



선입선출(FIFO) 대치 알고리즘의 실행

페이지 대치 알고리즘 - FIFO

- 문제점
  - 벨레디의 변이(Belady's Anomaly) 발생
    - 할당되는 프레임의 수가 증가해도 페이지 부재율이 증가하는 현상
    - 이로 인해 최적 페이지 대치 알고리즘을 추구하는 경향 발생
  - 아래의 참조 문자열을 적용하여 문제점 확인

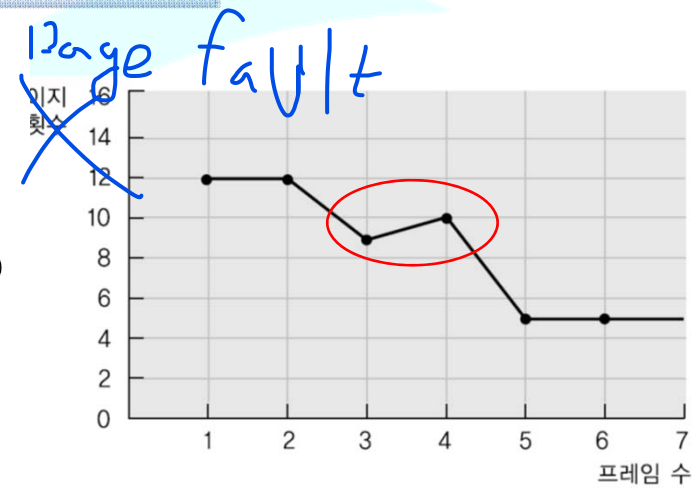
참조문자열 : 0,1,2,3,0,1,4,0,1,2,3,4

모든 페이지 프레임은 초기에 비어 있음

	0	1	2	3	0	1	4	0	1	2	3	4
3페이지 프레임	0	1	2	3	0	1	4	4	4	2	3	3
		0	1	2	3	0	1	1	1	4	2	2
			0	1	2	3	0	0	0	1	4	4
	P	P	P	P	P	P	P			P	P	← 페이지 부재 표시(9회)

	0	1	2	3	0	1	4	0	1	2	3	4
4페이지 프레임	0	1	2	3	3	3	4	0	1	2	3	4
		0	1	2	2	2	3	4	0	1	2	3
			0	1	1	1	2	3	4	0	1	2
				0	0	0	1	2	3	4	0	1
	P	P	P	P			P	P	P	P	P	P ← 페이지 부재 표지(10회)

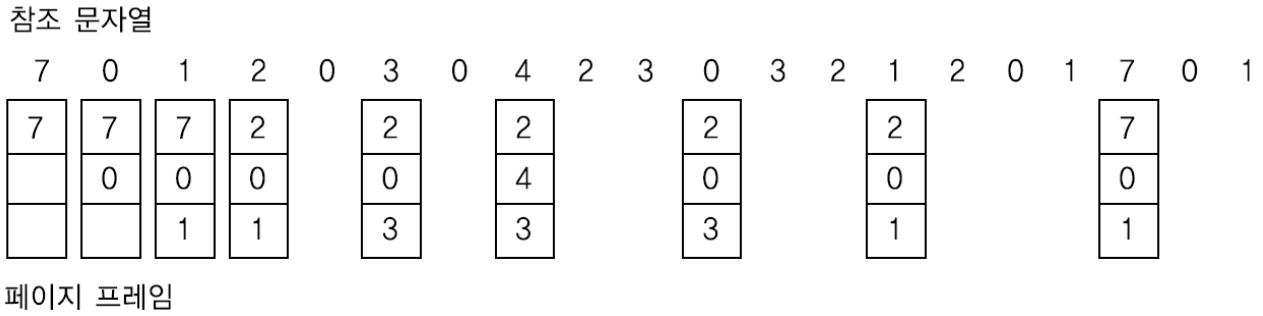
프레임별 실행결과



벨레디의 변이 현상

페이지 대치 알고리즘 – OPT

- 최적(Optimal) 페이지 대치 알고리즘
  - 모든 알고리즘 중 페이지 부재율이 가장 낮음
    - '앞으로 가장 오랜 기간 동안 사용하지 않을 페이지를 대치하라'는 사상을 표현
    - 고정된 프레임 수에 대해 가능한 가장 낮은 페이지 부재율이 보장됨
    - 최적 페이지 대치 알고리즘
      - 앞의 참조 문자열을 적용한 경우, 총 9번의 페이지 부재가 발생함



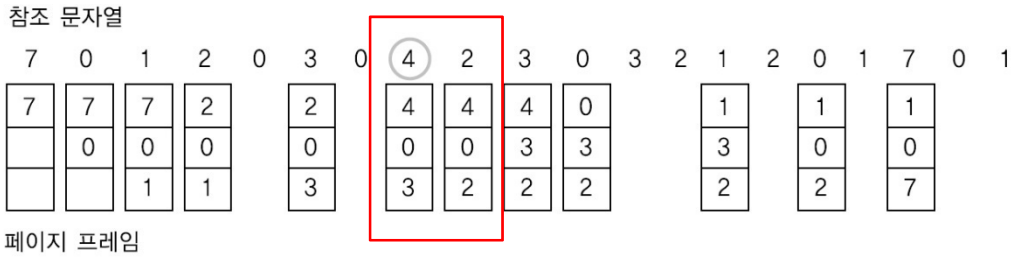
최적 페이지 대치 알고리즘

※ 현실적인 구현이 어려우며, 비교 연구를 위해 주로 사용됨

- 참조 문자열이 언제 사용될 것인가에 대한 정확한 정보를 요구함

페이지 대치 알고리즘 - LRU

- 최근 최소사용(LRU, Least Recently Used) 알고리즘
  - 과거의 데이터를 이용, 미래를 예측하기 위한 통계적 개념
    - 메모리의 지역성을 이용한 알고리즘으로 각 페이지에 마지막으로 사용된 시간을 연관시킴
    - 페이지 대치 시, 오랫동안 사용하지 않은 페이지를 선택
    - 최근 최소사용(LRU) 알고리즘
      - 앞의 참조 문자열 적용 시, 12번의 페이지 부재를 일으킴
      - 처음 5번의 부재 처리 과정은 최적 대치 알고리즘과 같으나, 이후 페이지 4에 대한 참조가 일어날 때 메모리 속 페이지 중 가장 늦게 사용한 페이지 2를 선택하여 대치함
      - 만약 페이지 2에 부재 발생 시, {0, 3, 4} 중 가장 늦게 사용한 페이지 3으로 대치함

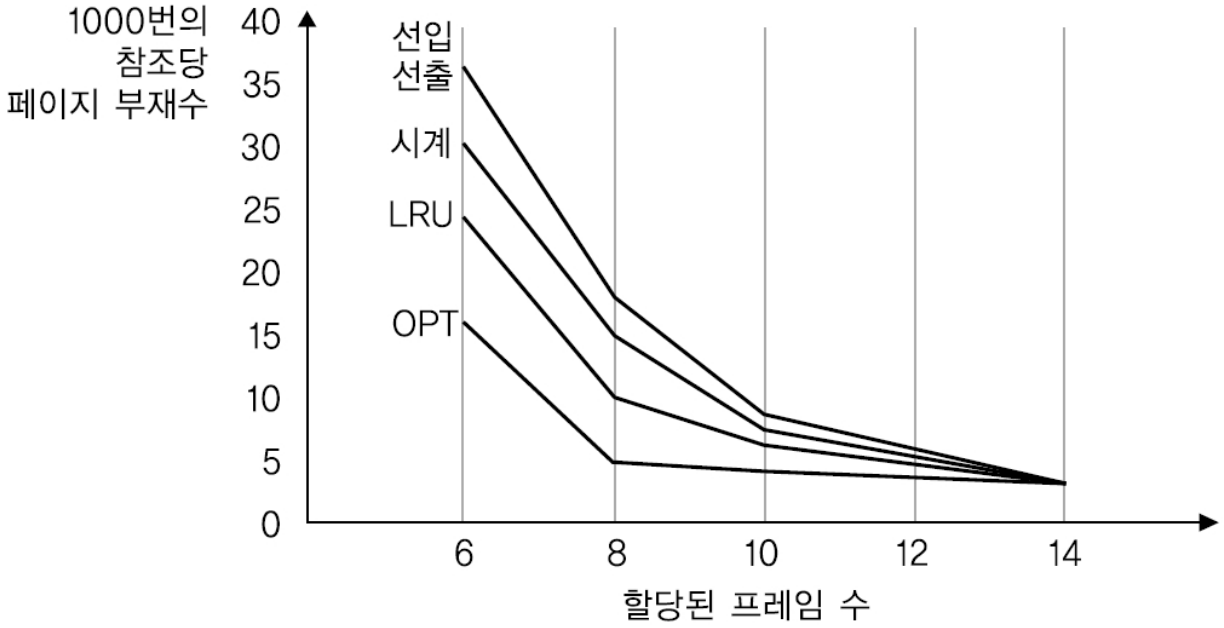


최근 최소사용(LRU) 알고리즘

※ 알고리즘의 구현을 위해 하드웨어 지원이 필요함

페이지 대치 알고리즘

- [참고] 알고리즘의 비교 분석
  - 배르(BAER, 1980년) 발표
    - 페이지 대치 알고리즘의 비교
      - 분석에 사용한 페이지 크기는 256word이며, 프레임 수를 6, 8, 10, 12, 14개로 변경시키면서 실행
      - 적은 수의 프레임을 사용할 경우 차이가 크게 나타남



페이지 대치 알고리즘의 비교

예측 페이징

- 예측 페이징(프리페칭, 프리페이징)
  - 프로세스들이 요구하는 페이지들을 예측, 메모리 공간에 여유가 있을 때 미리 로드
  - 예측 페이징 전략의 성공 요인
    - 예측 페이징 할당량 – 예측 페이징에 할당된 메인 메모리의 양
    - 한 번에 미리 로드된 페이지 수
    - 정책 – 어떤 페이지를 미리 로드할지 결정하는 경험적 지식
      - 즉, 시간적/공간적 지역성에 따라 예측
  - 신중한 설계 필요
    - 너무 많은 자원을 사용하거나 프로세스가 사용할 페이지를 잘못 결정할 경우, 요구 페이징 시스템보다 성능 저하
    - 해결책
      - 프로세스의 가상 주소 공간에 있는 페이지들을 2차 저장소에서도 그룹화