페이징 메모리 관리

물리 메모리= 메인 메모리

페이징

: 프로세스의 주소 공간을 페이지라고 불리는 고정 크기로 나누고 물리 메모리 역시 페이지와 동일한 크기의 프레임으로 나눈다. 프로세스의 각 페이지는 임의의 물리 프레임에 적재된다.

= 한 프로세스의 논리 주소 공간을 동떨어진 공간들에 배정할 수 있도록 지원

- 연속 할당에서처럼 연속적인 메모리 공간을 찾거나 만들 필요가 없다
- 외부 단편화 문제를 해결할 수 있는 메모리 관리 기술
- 동적 할당의 한 형태 → 모든 논리 주소는 페이지 하드웨어에 의해 물리 주소로 매핑

페이지 프레임 : 물리 메모리를 고정된 크기의 블록으로 분할

페이지 : 논리 주소 공간을 하나의 프레임과 같은 크기의 블록으로 분할

프레임 : 전체 메모리를 일정하게 나눈 것

운영체제는 모든 free frame들을 관리

프레임 테이블: 각 프레임의 할당 정보 기록

n page 크기의 프로그램 실행 위해서는 n개의 free frame을 찾고, 그곳에 적재함

논리 주소를 물리 주소로 변환하기 위해 페이지 테이블에 관련 정보 기록

- **페이지 테이블**은 프로세스마다 만들어지며, 프로세스에 속한 모든 스레드는 실행 시 프로세스의 페이지 테이블을 이용함. → PTBR이 가르키는 메인 메모리에 저장하여 존재함.
  - 메모리에서 각 페이지가 점유하는 주소를 가짐
  - 자연히 사용자 프로세스는 자신에게 할당되지 않은 메모리에 접근 불가

- 페이지의 물리 주소와 페이지 크기, 페이지의 논리 주소, 그리고 프로세스 번호
- 프로세스의 사용자 공간과 커널 공간 모두에 대해 적용됨

페이징의 우수성

1. 구현이 쉽다. → 고정 크기로 분할하기 때문
2. 이식성이 높다. → CPU에 의존 안 함
3. 융통성이 높다. → 시스템에 따라 페이지 크기를 다르게 설정
4. 메모리 활용과 시간 오버헤드 면에서 훨씬 우수하다.

   → 외부 발생 안 함, 내부 발생하지만 매우 작음, 홀 선택 알고리즘 실행 필요 없음

페이지의 크기

: 크기는 2의 제곱으로 증가 = 4KB

내부 단편화 생길 수 있음 (메모리 평균 크기 = 프로세스 당 1/2)

- 내부 단편화 감소를 위해 페이지 크기를 줄어야 함 → 페이지 테이블의 크기가 커져 공간 낭비됨
- 디스크 입장에선 페이지 크기가 클수록 효율적임 → 입출력 속도를 높이기 위해

커널 코드도 논리 주소로 되어 있으며, **시스템 호출**을 통해 페이지 K(커널 공간)의 커널 코드가 실행될 때 현재 프로세스의 페이지 테이블을 이용하여 물리 주소로 변환된다.

- MMU 장치는 페이지 테이블을 이용하여 논리 주소를 물리 주소로 변환함

페이지 크기가 크면, 내부 단편화도 커짐 하지만 미미하다.

프로세스를 구성하는 페이지 개수가 작아지고, 따라서 페이지 테이블의 크기도 작아진다.

페이지 크기가 크면 페이지 테이블 크기가 줄어듦. 그치만 입출력 속도를 높이기 위해 커지는 추세

페이징 개념 확인

프레임의 크기= 페이지의 크기

페이지 테이블의 항목 개수 = 페이지 개수

프레임의 개수 = 메모리 전체 크기/ 한 프레임 크기

페이지의 개수 = 프로세스 주소 공간/ 한 페이지의 크기

페이지 테이블 크기 = 페이지의 개수 X 페이지 테이블 항목의 크기

단편화 메모리의 평균 크기 = 한 페이지의 절반

페이징의 논리 주소

논리 주소 =[페이지 번호(p) , 옵셋(offset)]

- 페이지 테이블은 물리 메모리에 저장된다는 사실을 잊지마라탕
- 하드웨어 지원
  - CPU는 PTBR 제공 → 운영체제에 의해 제어됨 (메인 메모리에 저장)
  - MMU 장치 → 페이지 테이블을 참조하여 논리 주소의 물리 주소 변환, 저장/검색하는 캐시
- 운영체제 지원
  - 페이지 테이블 관리 기능 제공 → 할당된 페이지 테이블과 빈 프레임 리스트 생성 관리 유지
  - 프로세스 생성/소멸에 따라 동적으로 프레임 할당 및 반환 처리 → 페이지 테이블 갱신
  - 컨텍스트 스위칭 (재실행 → 재설정) 시 PTBR 로딩

페이징의 구현

운영체제는 페이지 테이블 관리 기능 제공

- 물리 메모리에 할당된 페이지 테이블과 빈 프레임 리스트 생성 관리 유지
- 프로세스 생성/소멸에 따라 동적으로 프레임 할당 및 반환 처리 → 페이지 테이블 갱신
- 컨텍스트 스위칭 시(재설정) PTBR 로딩

페이지 테이블 사용 시 문제점

1. 한 번의 메모리 조작을 위해 두 번의 메모리 접근 발생 → 물리 메모리의 액세스 횟수

   (메모리 액세스= 페이지 테이블 항목 읽기 1번+ 데이터n[i] 액세스 1번)

   → 실행 속도 저하

   문제 해결 방법 → TLB 사용으로 해결 [페이지 번호, 프레임 번호]

   : 특수한 고속의 하드웨어 캐쉬를 이용해 메모리 접근 문제를 해결 → 주소 변환 캐시로 불림

   TLB의 역할 : 논리 주소의 빠른 물리 주소 변환

   - 연관 메모리 사용 Key+Value → 모든 항목과 동시에 비교하여 단번에 프레임 번호를 출력
     - 페이지 번호 O → 프레임 번호 바로 알려줌

       X → 페이지 테이블에서 찾아 캐시 메모리 추가하고(교체 작업) 다시~
   - 처음에 TLB 미스가 발생하는 한 번의 경우에만 물리 메모리를 2번 액세스함
   - 동일한 페이지를 연속하여 액세스하는 동안 TLB 히트가 계속 발생 → 횟수 줄고 → 성능 향상
2. 페이지 테이블의 낭비 → 프로세스의 실제 크기는 매우 작아 대부분의 테이블 항목이 비어있음

TLB와 참조의 지역성

CPU 패키지의 MMU 장치 내에 존재함

- TLB는 참조의 지역성으로 인해 효과적인 전략임
- TLB 사용 시 순차 접근 시 실행 속도 빠름
- TLB 사용 시 임의 접근이나 반복이 없는 경우엔 느림 → TLB miss로 TLB 항목 잦은 교체 발생

TLB 성능

- TLB 히트율을 높이기 위해선 TLB 항목 수 늘리기 → 비용도 상승됨 (단점)
- 페이지가 클수록
  - TLB 히트 증가 → 실행 성능 향상

    - 내부 단편화 증가 → 메모리 낭비
- TLB 도달 범위 : 모든 항목이 채워졌을 때, 미스 없이 접근할 수 있는 메모리 접근 범위
    - TLB 항목 수 x 페이지 크기

TLB를 고려한 컨텍스트 스위칭 재정립 과정

1. CPU의 모든 레지스터를 PCB에 저장
2. 새로 스케줄된 프로세스의 PCB에 저장된 페이지 테이블의 주소를 CPU의 PTBR에 적재
3. TLB 캐시의 모든 항목을 지움 → 이전 프로세스의 잔재이므로
4. 새 프로세스의 PCB에 저장된 레지스터 값들을 CPU에 적재 후 실행 재개

   → 이후 TLB miss가 발생하며 TLB 항목이 채워져 감

페이지 테이블의 메모리 낭비

- 페이지 테이블의 일부 항목만 사용
- 프로세스마다 페이지 테이블 존재

페이지 테이블 낭비 문제의 해결책

역 페이지 테이블 IPT

: 프레임을 중심으로 물리 메모리의 전체 프레임에 대해 각 프레임이 어떤 프로세스의 어떤 페이지에 할당되었는지 나타내는 테이블로 시스템에 1개만 둔다.

- 물리 주소 공간에 대응시켜 구성 (일반적으로는 프로세스마다 논리 주소 공간에 대응함)
- 각 항목은 메모리의 각 프레임에 대응 (시스템 전체에 단 하나의 페이지 테이블만 존재)
    - 역 페이지 테이블 항목의 수 = 프레임의 수 = 물리 메모리의 크기/ 프레임의 크기
- 논리주소 형식 = [프로세스 번호(PID), 페이지 번호(p), 옵셋(offset)]
- 항목 : 프로세스 번호, 페이지 번호

- 특징

- 페이지 테이블 크기를 획기적으로 줄일 수 있음
- 특정 페이지 프레임을 탐색하기 위해 더 많은 시간 소요 → 페이지 테이블 전체를 검색해야함
- 탐색 시간을 줄이기 위해 해시 테이블 사용

  ⇒ 3번의 메모리 조작 발생 (hash table → inverted page table → 주기억장치)

  ⇒ 연관 메모리 사용 → 단점: 고가의 비용

멀티 레벨 페이지 테이블

- 논리 주소 공간을 여러 단계의 페이지 테이블로 나눔
  - 대용량의 페이지 테이블을 주기억장치처럼 취급하여 다시 페이징 함
- **2-레벨 페이징 기법**
  - 페이지 테이블을 다시 페이징 → 논리 주소의 페이지 번호 부분을 2개의 레벨로 나눔
  - [페이지 디렉터리 인덱스, 페이지 테이블 인덱스, 옵셋]

세그멘테이션

: 가변 길이의 연관된 정보들의 집합

- 각 세그먼트들은 이름과 길이를 가짐 → 외부 단편화 발생

: 사용자 관점에서의 메모리 관리 기법

- 논리 주소 공간을 세그먼트들의 집합으로 정의
  - 코드, 데이터, 스택, 힙 세그먼트
  - 컴파일러, 링커, 로더, 운영체제에 의해 관리됨
- 세그먼트 테이블은 시스템에 1개만 있음
- 하드웨어의 지원
  - CPU STBR : 세그먼트 테이블에 대한 메모리상의 시작 주소
  - MMU: 논리 주소를 물리 주소로 매핑
  - 세그먼트 테이블 : 각 세그먼트 별로 시작 주소와 길이 정보 유지

- 운영체제의 지원
  - 세그먼트의 동적 할당/반환 및 세그먼트 테이블 관리 기능 구현
  - 프로세스 생성/ 소멸에 따라 동적으로 할당/반환
  - 물리 메모리에 할당된 세그먼트 테이블과 자유 공간에 대한 자료 유지
  - 컨텍스트 스위칭 때 CPU 레지스터에 적절한 값 적재
  - 컴파일러, 링커, 로더의 지원
- 세그먼트 테이블의 항목은 어떤 정보로 구성되는가?
  - 세그먼트의 크기와 세그먼트의 물리 주소
    - 크기 정보가 저장되는 목적 : CPU에 의해 발생된 논리 주소가 세그먼트가 할당된 물리 메모리의 영역을 넘어섰는지 판단하기 위해

- 세그먼테이션보다 페이징이 더 우수하다