메모리 관리 2

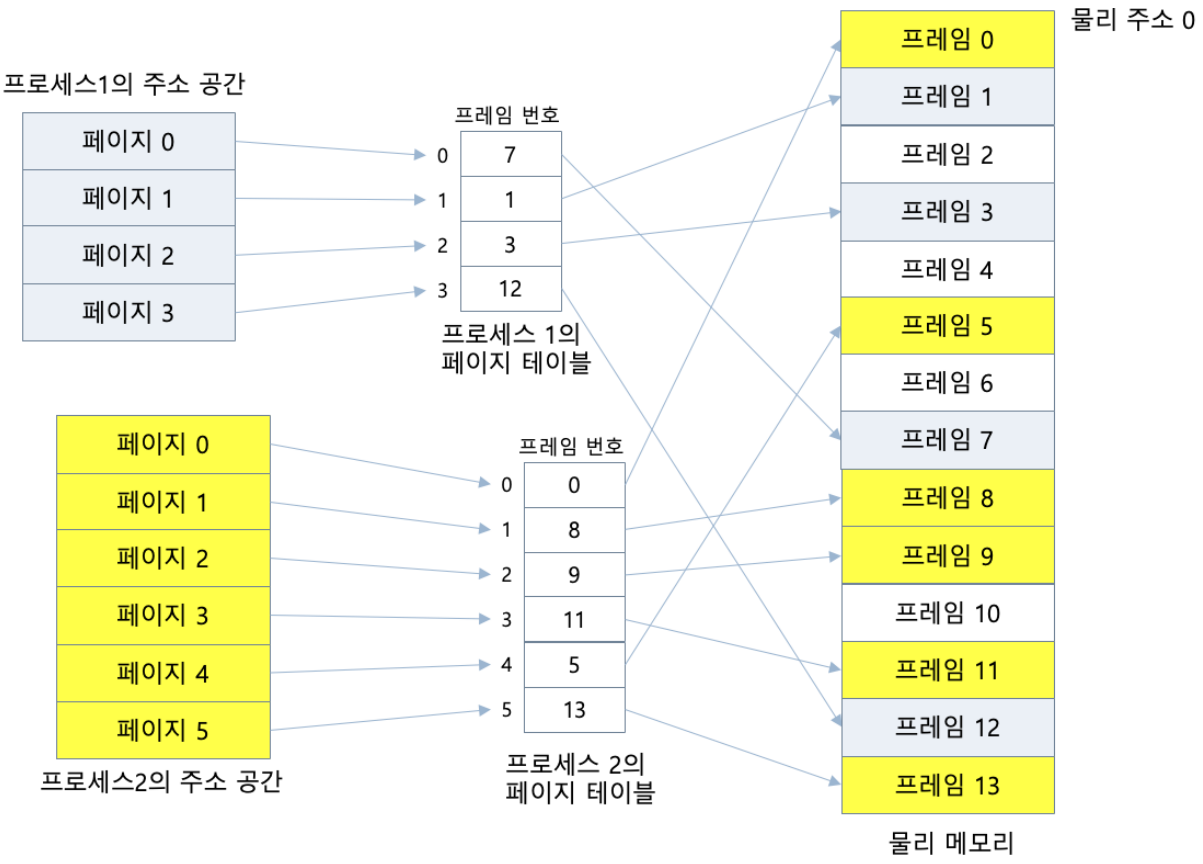
Paging

페이징

- 프로세스의 주소 공간을 0번지부터 동일한 크기의 페이지로 나눔
- 물리 메모리 역시 0번지 부터 페이지 크기로 나누고, 프레임이라고 부른다.
- 코드, 데이터, 스택 등 프로세스의 구성 요소에 상관없이 고정 크기로 분할한 단위임
- 페이지와 프레임에 번호를 붙여 관리한다. → 페이지 테이블
    - 각 페이지에 대해 페이지 번호와 프레임 번호를 1:1로 저장하는 테이블

페이징 기법

- 프로세스의 주소 공간과 물리 메모리를 페이지 단위로 분할하고, 프로세스의 각 페이지를 물리 메모리의 프레임에 분산 할당하고 관리하는 기법
- 프로세스의 주소 공간
    - 선형적인 주소 공간(0에서 시작하여 연속적인 주소 공간)
- 프로세스마다 페이지 테이블 있음
- MMU에 의한 논리 주소의 물리 주소 변환
- 물리 메모리의 빈 프레임 리스트 관리 필요
  → 프레임 할당 알고리즘 : 빈 프레임 중에서 선택알고리즘 필요
- 내부 단편화가 발생한다.



페이징 기법 장점

- 메모리를 0번지부터 고정 크기로 단순하게 분할하기에 구현이 쉬움
- 페이징 메모리 관리를 위해 cpu에 의존하는 것이 아니기에 다른 컴퓨터 시스템에서도 사용가능
- 시스템에 따라 페이지 크기 달리 설정 가능
- 메모리 활용과 시간 측면에서 좋음
  - 외부단편화가 발생하지 않음
  - 내부단편화는 발생하지만 작음
  - 홀 선택 알고리즘 실행할 필요없음

그렇다면 페이지 테이블을 어떻게 구성하는가?

페이지 테이블 구현

- 페이지 테이블은 메인 메모리에 상주함

- page-table base register가 page table을 가르킨다. → 컨텍스트 스위칭의 경우 레스터의 내용만 변경하면 됨
- page-table length register가 테이블 크기를 보관한다.
- page table
  - protection bit : page에 대한 접근권한
  - valid-invalid bit : valid는 해당 주소의 frame에 프로세스를 구성하는 유효한 내용 존재

Translation Look-aside Buffer (TLB)

- 메모리 주소 변환을 위한 별도의 캐시 메모리
- 페이지 테이블에서 자주 참조되는 일부 엔트리를 캐싱하고 있음
- key-value 쌍으로 데이터를 관리하는 associative memory임
  - key에는 page number
  - value에는 frame number
- cpu는 page table 보다 tlb를 우선적으로 참조함
- parallel search
  - 전체를 탐색해서 원하는 엔트리를 찾는 것은 비효율적
  - tlb의 성능을 높이려면 page의 크기를 키우면 됨
- 컨텍스트 스위칭 경우
  - cpu의 모든 레지스터를 pcb에 저장
  - pcb에 있는 프로세스 페이지 테이블 주소를 mmu의 page table base register로 로딩
  - tlb 내용 모두 지움
  - 새 프로세스 컨텍스트를 pcb에서 cpu로 로딩한다.

페이지 테이블의 메모리 낭비

메모리 낭비 발생

- 32bit cpu 환경에서 프로세스 당 페이지 테이블 크기 4kb라고 가정한다면 2^32 / 2^12 = 2^20개의 페이지 구성
- 프로세스 당 페이지 테이블의 크기는 2^20 X 4바이트 = 4MB

해결책

1. 역페이지 테이블
2. 멀티 레벨 페이지 테이블
3. two-level 페이지 테이블
    a. 페이지 테이블들의 페이지 화
        i. 논리주소는 페이지 번호와 옵셋으로 구성한다.
        ii. 논리주소의 페이지 번호 부분을 2개의 레벨로 나눴음