9 메모리 관리3

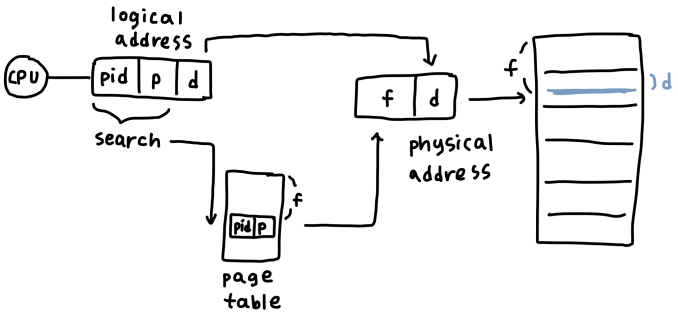
Inverted page table

역방향 페이지 테이블

- page table이 매우 큰 이유
  - 모든 process별로 그 logical address에 대응하는 *모든 page에 대해 page table entry가 존재*
  - 대응하는 page가 메모리에 있든 아니든 간에 page table에는 entry로 존재
- inverted page table

  → page frame 하나당 page table에 하나의 entry를 둔 것 (system - wide)
  - 각 page table entry는 각각의 물리적 메모리의 page frame이 담고 있는 내용 표시 (process id, process의 logical address)
  - 단점 : *테이블 전체를 탐색*해야 함 → associatice register 사용 (expensive)

    → page table 각 위치를 병렬적으로 한 번에 탐색할 수 있는 하드웨어



⇒ 물리적 frame 번호를 얻는 과정

- 물리적 메모리 *frame 하나하나당 page table entry 존재*
- **pid** : 누구의 논리적인 page 번호인지를 담음 → proces A에도 p번째 page 존재, process B도 존재하니까
- process마다 page table을 각각 두지 않고 하나의 page table로만 접근 가능 but 주소 변환에는 도움을 주지 않음

Shared page

: shared code를 담는 페이지

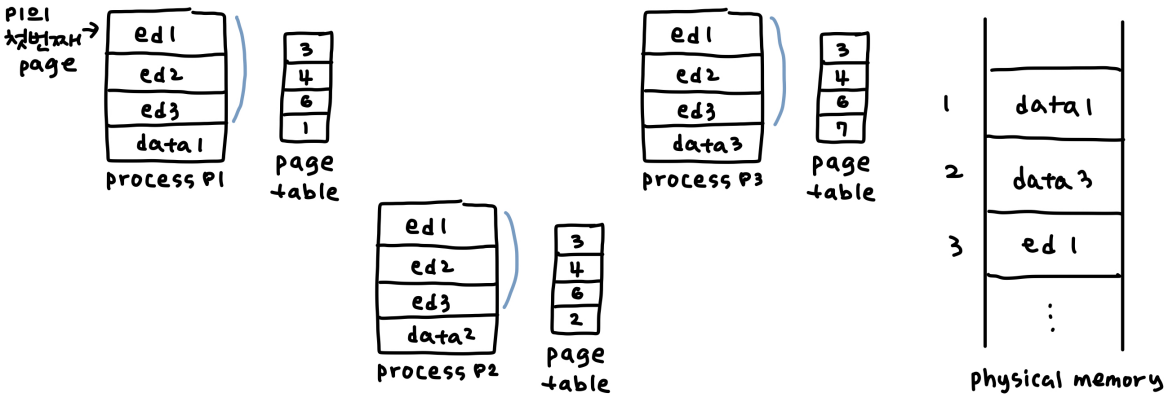
- **shared code**
  - re-entrant code (=pure code)
  - **read-only**로 하여 프로세스 간에 하나의 code만 메모리에 올림

    - shared code는 모든 프로세스의 logical address space에서 동일한 위치에 있어야 함 → page 번호가 같아야 함

  ⇒ 두 가지 제약 사항

- **private code and data**
    - 각 프로세스들은 *독자적으로 메모리에 올림*
    - private data는 logical address space의 아무 곳에 와도 무방

> 💡 shared memory와 다른 점
> ↓
> communication 목적으로 메모리의 일부를 공유 ⇒ read/write가 모두 가능



⇒ editor is shared

동일한 프로그램 3개를 돌리면 코드는 동일하고 data만 다르니까 코드를 공유코드로 둠

Segmentation

: 의미 단위로 잘라서 물리적 단위에 올림 → 크기가 각각 다름

프로그램은 의미 단위인 *여러 개의 segment로 구성*

- 작게는 프로그램을 구성하는 함수 하나하나를 세그먼트로 정의
- 크게는 프로그램 전체를 하나의 세그먼트로 정의 가능
- 일반적으로는 code, data, stack 부분이 하나씩의 세그먼트로 저의됨

⇒ segment는 다음과 같은 logical unit들임

main() / function / global variables / stack / symbol table / arrays

Segmentation architecture

- logical address → <segment number, offset>
- *segment-table*
  - **base** : 물리적 메모리에서의 *시작위치*

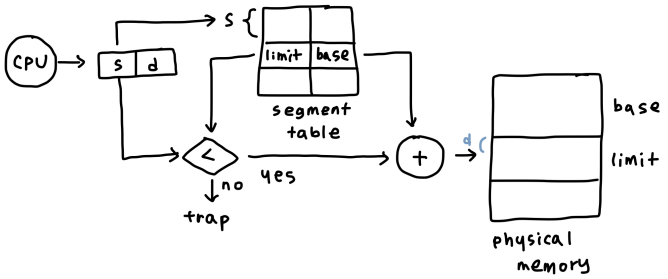
    ⇒ 크기가 segment별로 다르기 때문에 byte 단위
  - **limit** : *segment의 길이*

    ⇒ paging은 크기가 다 같아서 필요가 없지만 segment는 길이가 다 달라서 담고 있어야 함
- segment-table base register (STBR)

  물리적 메모리에서의 *segment table의 위치* → 시작 위치를 담고 있음
- segment-table length register (STLR)

  프로그램이 사용하는 *segment의 수* → segment table의 길이 = 몇 개의 segment로 구성되어 있는지

  → 존재하는 segment 수가 STLR보다 더 큰 segment번호면 접근 하려해도 trap



Protection

- 각 segment 별로 *protection bit*가 있음
- each entry
  - valid bit = 0 → illegal segment
  - read/write/execution 권한 bit

Sharing

- shared segment → 의미 단위라 paging보다 공유, 보안에 있어 효과적 ⇒ 의미 단위로 해야 하는 일들
- same segment number

Allocation

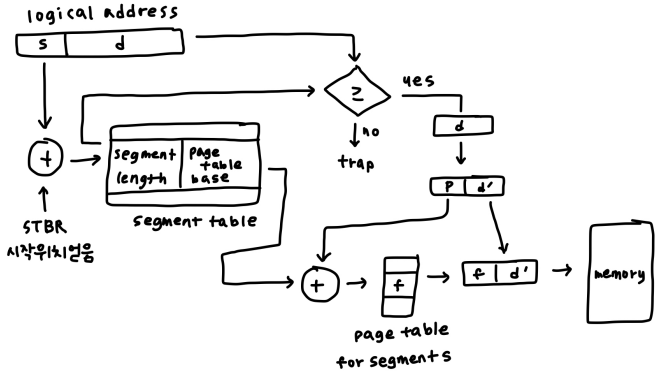
- first fit / best fit
- *external fragmentation* 발생 → 가변 분할 방식에서와 동일한 문제점

Segmentation with paging

→ segment가 여러 개의 page로 구성

- pure segmentation과의 차이점

  segment-table entry가 segment의 base address를 가지고 있는 것이 아니라 segment를 구성하는 page table의 base address를 가지고 있음



→ page 단위로 물리적 메모리에 올라감