Paging : Smaller Tables

Paging : Linear Tables

- 주로 프로세스 하나당 하나의 페이지 테이블을 관리
- 32bit address space / 4kb pages / 4-byte page table entry이라고 치면 Page Table Size = 2^32/2^12 * 4 = 4 MB
- 사이즈가 너무 커서 메모리를 너무 잡아먹음
- 32bit address space / 16kb pages / 4-byte page table entry이라고 치면 Page Table Size = 2^32/2^14 * 4 = 1 MB
- 페이지테이블이 너무 크면 Internal Fragmentation(필요한 것보다 훨씬 많은 공간이 할당됨) 발생 -> Page Table에 빈 Entry가 너무 많음

Hybrid Approach : Paging & Segments

- 페이지 테이블의 메모리 오버헤드를 줄이기 위해 사용
- Base를 세그먼트를 가리키는데 쓰는게 아니라, 해당 세그먼트의 페이지 테이블의 물리적 주소를 Hold하는데 씀
- Bounds는 Page Table의 끝을 가리키는데 씀 ex) 각각의 프로세스는 3개의 페이지 테이블을 가짐(Code, Stack, Heap) -> 프로세스가 실행중일때, 세그먼트들의 Base값은 해당 세그먼트의 페이지테이블의 물리적 주소를 Hold

TLB Miss 발생시

- 하드웨어가 페이지 테이블에서 물리적 주소를 받아오는 과정
    1. 하드웨어가 Segment Bits(SN)을 사용해서 어느 Base & Bounds 쌍을 사용할지 결정
    2. 해당 세그먼트의 페이지테이블에서 물리적 주소를 받아와서 VPN와 합쳐서 PTE에 해당하는 주소를 생산

Problem

- 엄청 크지만 드문드문 차 있는 Heap을 사용할경우, Page Table 낭비는 여전함
- External Fragmentation(구멍이 숭숭 나있는 경우) 발생

Multi-level Page Tables

- Linear Page Table을 Tree같은 형태로 바꿈
- 페이지 테이블을 작은 페이지 크기의 단위로 쪼갬
- 해당 페이지의 모든 PTE가 비어있는 경우, 그 페이지를 메모리에 할당 안 함
- 페이지 테이블의 페이지가 Valid한지 확인하기 위해 Page Directory라는 구조 사용

Page Directory Entries

- 페이지 디렉토리는 하나의 페이지테이블의 페이지당 하나의 Entry를 지님
- 이 Entry(페이지테이블의 페이지)를 PDE(Page Directory Entry)라고 부름
- PDE는 Valid bit와 PFN(Page Frame Number)를 지님

Advantage

- Address Space를 사용하고 있는 만큼만 page-table space를 사용함

- 운영체제가 필요할때 free page를 가져가서 할당 할 수 있음

Disadvantage

- time-space trade-off(즉 공간효율성은 좋아지지만 시간효율성이 안좋아짐)
- 복잡함

Level of Indirection

- Multi-level 구조는 페이지 디렉토리를 사용해서 Indirection의 레벨을 조정 할 수 있음
- Indirection은 페이지테이블의 페이지들을 물리적 메모리 아무곳에나 할당함

Example

- VPN / PTE per page가 Page Directory Index의 역할 -> PDE가 invalid하면 Exception throw -> PDE가 valid하면 해당 PTE를 찾아감

More than two level

- Page Directory Index부분이 지나치게 길어지면, Tree의 레벨을 하나더 늘린다 -> 반으로 갈라서 page directory의 여러 page로 만든다

Inverted Page Tables

- 하나의 페이지 테이블을 가짐(물리적 페이지 하나가 Entry 하나를 구성)
- Entry는 어떤 프로세스가 해당 페이지를 쓰고있는지, 어느 가상 페이지가 해당되는지를 저장