09. Non-continuous Allocation / Virtual Memory

Virtual Storage (Memory)

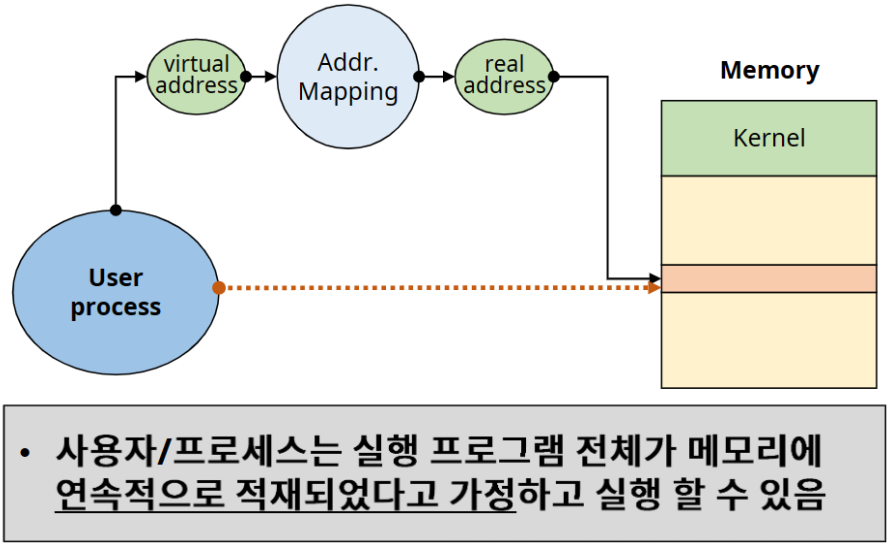
- Non-continuous Allocation
  - 사용자 프로그램을 여러 개의 연속되지 않은 블록으로 분할
  - 실행 시, 필요한 블록만 메모리에 적재, 나머지 블록은 Swap Device에 존재
- Virtual Storage 기법
  - Paging System
  - Segmentation System
  - Hybrid Paging/Segmentation System

Address Mapping

Continuous Allocation

- Relative Address (상대 주소)
  - 프로그램 시작 주소를 0으로 가정한 주소
- Relocation (재배치)
  - 메모리 할당 후, 할당된 주소에 따라 상대 주소를 조정하는 작업
- Address Mapping (Continuous Allocation)
  - 상대주소를 물리주소로 매핑하는 작업, 과정에 재배치 이루어짐

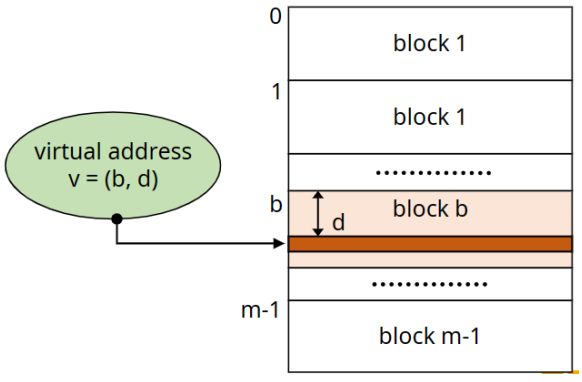
Non-continuous Allocation



- Virtual Address (가상 주소): relative address와 비슷한 개념
  - logical address (논리 주소)
  - 연속된 메모리 할당을 가정한 주소
- Real Address (실제 주소) = absolute (physical) address
  - 실제 메모리에 적재된 주소
- Address Mapping (Non-continuous Allocation)
  - 가상 주소를 실제 주소로 바꿔주는 것

Block Mapping

[정의]



- 간단한 Address Mapping 기법
- 사용자 프로그램을 블록 단위로 분할/관리
  - 각 블록에 대한 address mapping 정보 유지

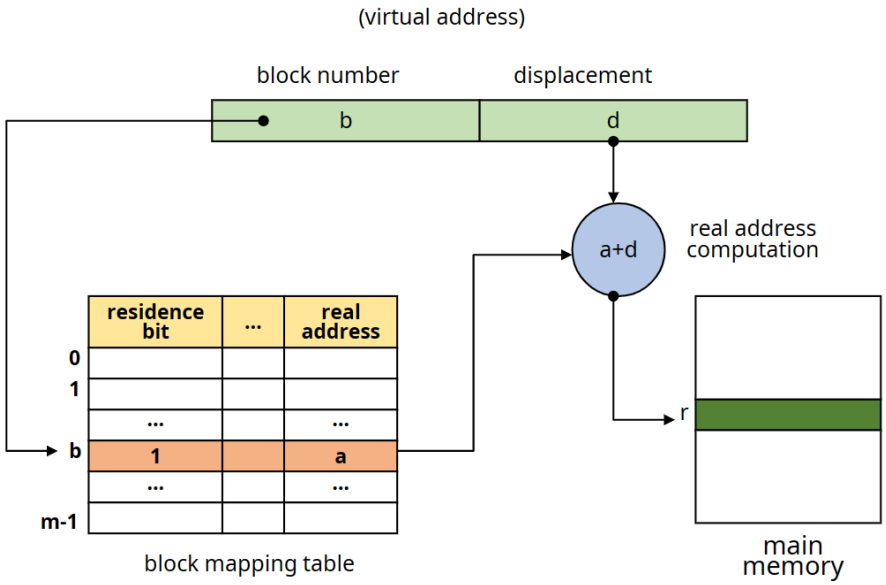
- Virtual address : `v = (b,d)`
  - `b` = block number
  - `d` = displacement(offset) in a block

[Block Map Table]

block number	residence bit	...	real address
0 1 2 ...		⋮	
b	1	.....	a
... m-1		⋮	

- Address Mapping 정보 관리
  - 커널 공간에 **프로세스마다 하나**의 BMT의 가짐
- Residence bit: 해당 블록이 메모리에 적재되었는지 여부 (0/1)

[과정]

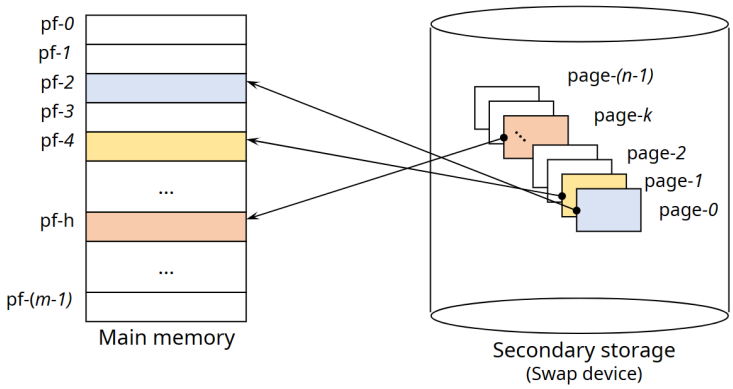


1. **프로세스의 BMT에 접근**
2. **BMT에서 block b에 대한 항목(entry)를 찾음**
3. **Residence bit 검사**
   - ① Residence bit = 0 경우,
     swap device에서 해당 블록을 메모리로 가져 옴
     BTM 업데이트 후 3-② 단계 수행
   - ② Residence bit = 1 경우,
     BMT에서 b에 대한 real address 값 a 확인
4. **실제 주소 r 계산 (r = a + d)**
5. **r을 이용하여 메모리에 접근**

Virtual Storage Methods

Paging System

> 프로그램을 같은 크기의 블록(Page)으로 분할



- Page
  - 프로그램의 분할된 블록
- Page Frame
  - 메모리의 분할 영역. Page와 크기 동일

특징

- 논리적 분할이 아님. 크기에 따른 분할
  - Page를 공유하고 보호하는 과정이 Segmentation 대비 복잡함
- Page 크기로 나눠놨기 때문에 Segmentation 대비 simple & efficient
- No external fragmentation: Page에 올라가거나 안 올라가거나 두 경우만 존재

- internal fragmentation 발생 가능

Address Mapping

- Virtual address : v = (p,d)
  - p: page number
  - d: displacement(offset)
- Address Mapping 자료 구조
  - Page Map Table 사용
- Address Mapping Mechanism
  - Direct Mapping (직접 사상): block mapping과 유사
  - Associative Mapping (연관 사상)
    - TLB (Translation Look-aside Buffer)
  - Hybrid direct/associative mapping

Page Map Table (PMT)

page number	residence bit	secondary storage address	other fields	page frame number
0	1	S_0	· · ·	2
1	1	S_1	· · ·	4
2	0	S_2	· · ·	-
· · ·	· · ·	...	· · ·	· · ·
k	1	S_k	· · ·	h
· · ·	· · ·	...	· · ·	· · ·
n-1	0	S_{n-1}	· · ·	-

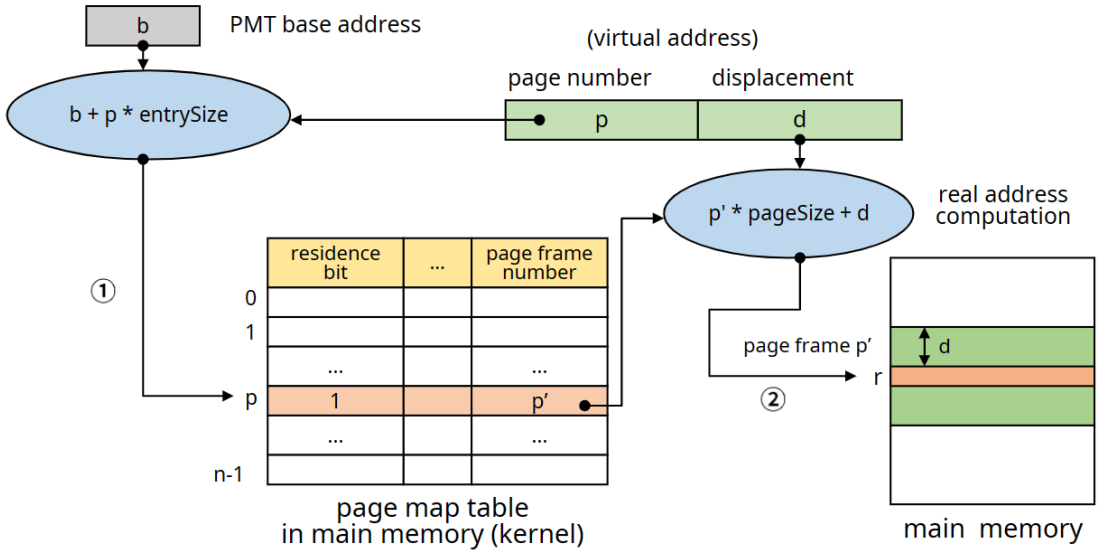
- Secondary storage address: swap device 상 어떤 페이지에 올라와 있는지

Address Mapping Mechanism: Direct Mapping

[개요]

- Block Mapping 방법과 유사
- 가정
  - PMT를 커널 안에 저장

[과정]



1. **해당 프로세스의 PMT가 저장되어 있는 주소 b에 접근**
2. **해당 PMT에서 page p에 대한 entry 찾음**
   - p의 entry 위치 = b + p * entrySize
3. **찾아진 entry의 존재 비트 검사**
   - ① Residence bit = 0 인 경우 **(page fault)**, swap device에서 해당 page를 메모리로 적재 PMT를 갱신한 후 3-② 단계 수행
   - ② Residence bit = 1인 경우, 해당 entry에서 page frame 번호 p'를 확인
4. **p'와 가상 주소의 변위 d를 사용하여 실제 주소 r 형성**
   - r = p' * pageSize + d
5. **실제 주소 r로 주기억장치에 접근**

Context switching 발생 (I/O)
➔Overhead

- Page Fault
  - Residence bit = 0인 경우. 페이지 읽는 데 실패
  - Swap device에서 해당 Page를 메모리에 적재할 때 (running → asleep: context switch 발생, overhead 큼)
- 문제점
  - 메모리 접근 횟수가 2배: 성능 저하
    - 커널의 PMT를 보기 위해 1번, 실제 주소 접근하기 위해 1번
  - PMT를 위한 메모리 공간 필요

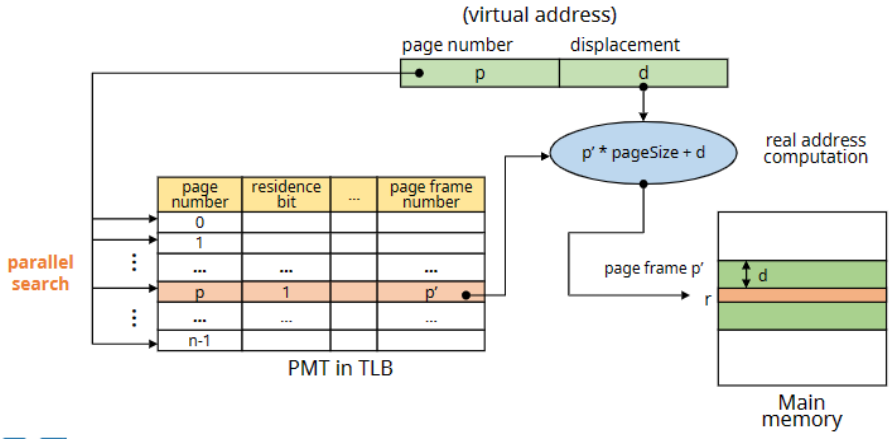
- 해결 방안
  - Associative Mapping (TLB)
  - PMT를 위한 전용 기억장치(공간) 사용: Dedicated register or Cache memory
  - Hierarchical paging, Hashed page table, Inverted page table 등

Address Mapping Mechanism: Associative Mapping

[개요]

- TLB에 PMT 적재 후, PMT 병렬 탐색
  - TLB: Associative high-speed memory
- low overhead, high speed

[과정]



- page number만 주면, TLB를 병렬 탐색해 real address 계산

[단점]

- TLB가 비쌈. 큰 PMT 다루기 어려움

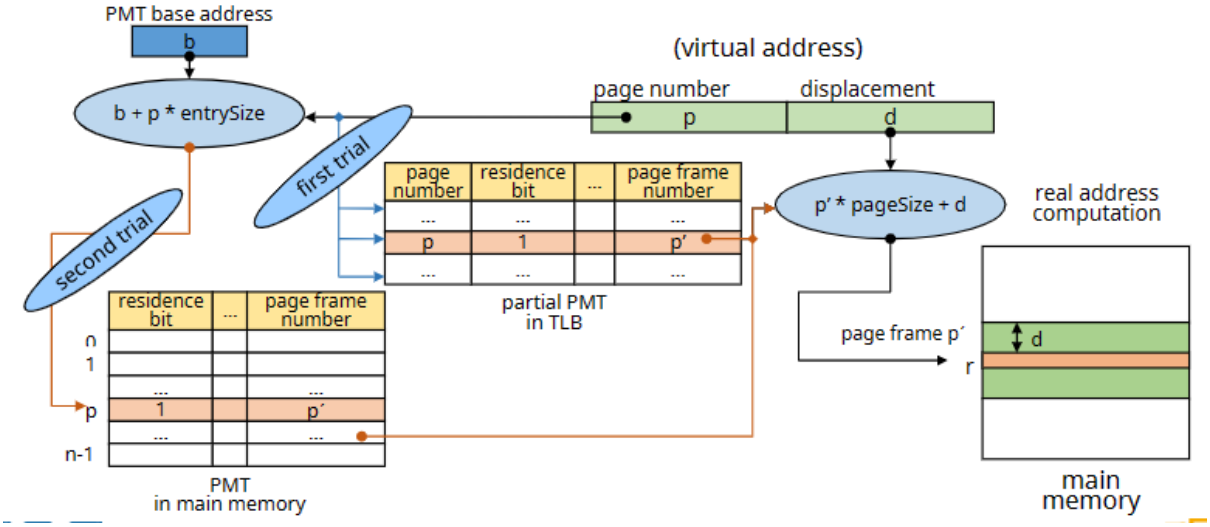
Address Mapping Mechanism: Hybrid Direct/Associative Mapping

[개요]

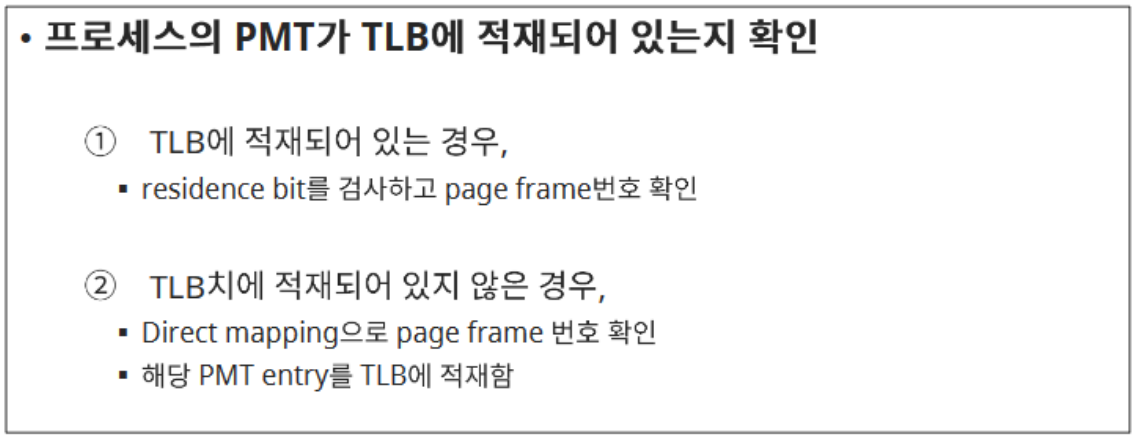
- 두 기법을 혼합하여 사용
  - HW 비용은 줄이고, Associative Mapping의 장점 활용
- 작은 크기의 TLB를 사용

- PMT: 메모리(커널 공간)에 저장
- TLB: PMT 중 일부 entry를 적재. 최근에 사용된 페이지에 대한 entry 저장
  - locality 활용: temporal, spatial locality

[과정]



- 다음 과정이 추가됨

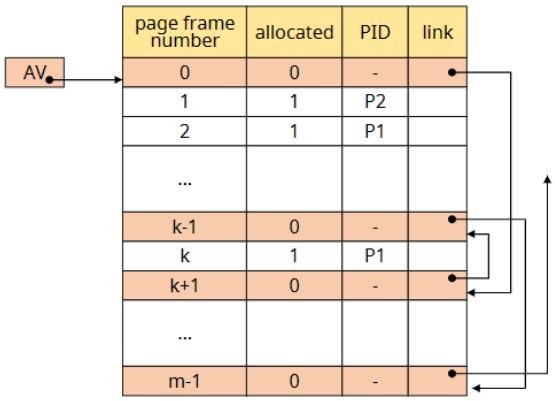


1) Associative Mapping 2) Directive Mapping + TLB에 적재

Memory Management

- Page와 같은 크기(Page Frame)로 메모리를 미리 분할해 관리/사용
- Frame Table
  - Page frame 당 하나의 entry

[Frame Table]



- Allocated/available field: 0/1
- PID field
- Link field: for free list (사용 가능한 fp를 연결 리스트로 저장)
- AV: free list header (free list의 시작점)

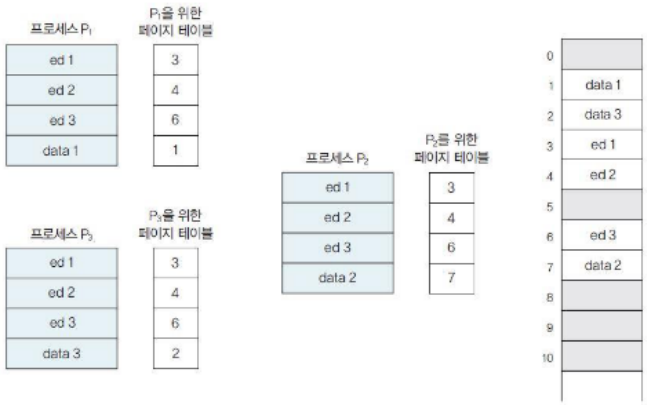
Page Sharing

- Paging System에서는 여러 프로세스가 특정 page를 공유 가능
  - Non-continuous allocation이기 때문에 가능

[공유 가능 Page]

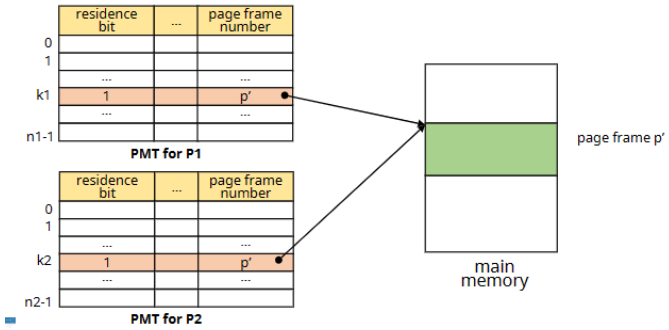
- Procedure (aka function) pages
  - Pure code
- Data page
  - read-only data
  - read-write data: 병행성 제어 기업 관리 하에서만 가능

[예제: 특정 프로그램을 3인이 사용할 때]



- 공통적으로 사용하는 프로그램 코드 부분은 한 번만 올려놓음. 데이터만 별도 관리

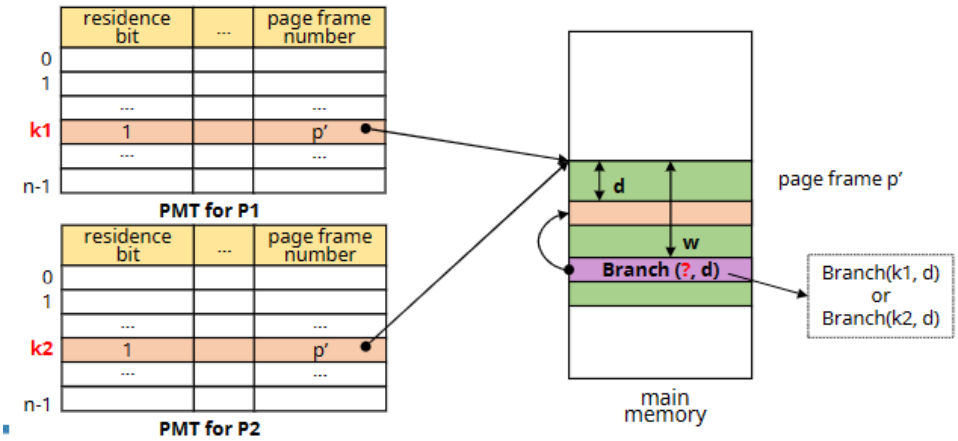
[Data Page Sharing]



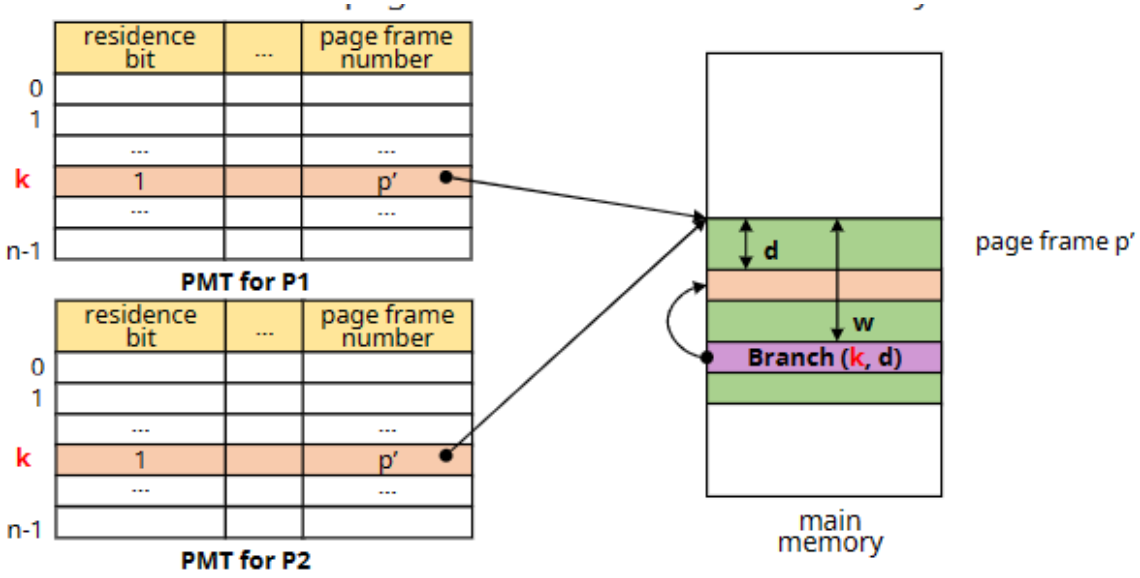
- 다른 PMT가 있어도 동일한 Page frame number로 공유할 수 있음

[Procedure Page Sharing]

- 문제
  - 다른 PMT를 사용할 때, 다른 주소 같은 Page frame number를 가리킨다면, offset으로 주소 계산할 때 서로 다른 실제 주소로 갈 수 있음

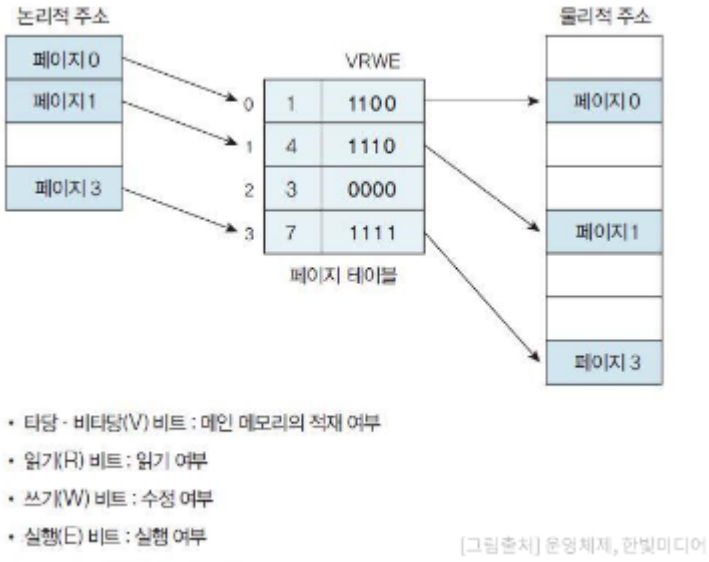


- 해결책
  - 프로세스들이 shared page에 대한 정보를 PMT와 같은 entry에 저장하도록 함



Page Protection

- 여러 프로세스가 page를 공유할 때, Protection bit을 사용해서 VRWE 권한 관리
  - 프로세스가 해당 페이지에 대해 어떤 작업을 할 수 있는지



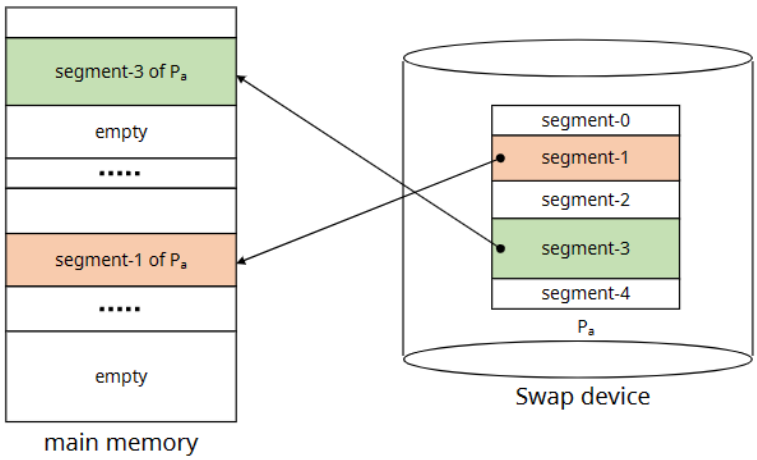
Paging System: Summary

- 분할 방식
  - 프로그램을 고정된 크기의 block으로 분할(page)

- 메모리를 block size로 미리 분할(page frame)

- 특징
  - 외부 단편화 X
  - 메모리 통합/압축 필요 X
  - 프로그램의 논리적 구조 고려 X. Page sharing/protection 복잡
- 필요한 page만 page frame에 적재해 사용 ⇒ 메모리의 효율적 활용
- Page Mapping Overhead
  - 페이지 매핑을 위해 메모리 공간 및 추가 메모리 접근 필요
  - 전용 HW(eg. TLB) 활용으로 해결 가능 but 하드웨어 비용 증가

Segmentation System



논리적 분할했기 때문에 Segment 크기 다름

- 프로그램을 논리적 블록으로 분할 (Segment)
  - 블록의 크기가 서로 다를 수 있음
  - ex) stack, heap, main procedure, shared library, etc
- 특징
  - 미리 메모리 분할 X. Variable Partition Multiprogramming과 유사
  - Segment sharing/protection 용이
  - Address mapping 및 메모리 관리의 overhead 큼
  - No internal fragmentation. External Fragmentation 발생 가능

Address Mapping

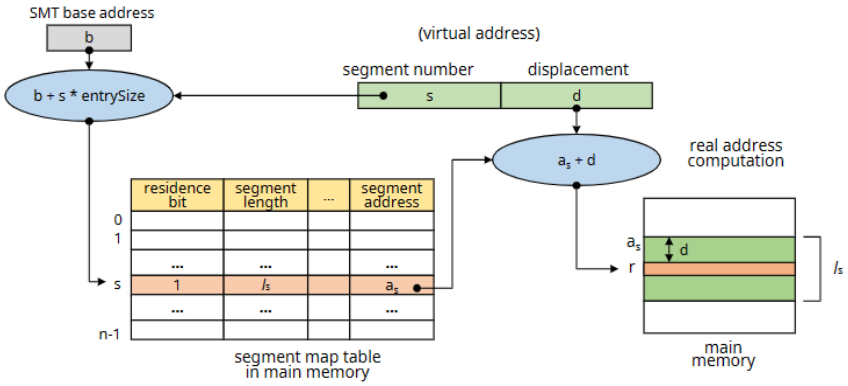
- Virtual address: `v = (s,d)`
  - s: segment number
  - d: displacement in a segment
- Segment Map Table (SMT)
- Address Mapping Mechanism
  - Paging System과 유사

Segment Map Table

segment number	residence bit	secondary storage address	**segment length**	**protection bits (R/W/X/A)**	other fields	segment address in memory
0	1	S_0	l_0	RW	...	a_0
1	1	S_1	l_1	RW	...	a_1
2	0	S_2	l_2	RX	...	-
...	...	...	...	...	...	...
k	1	S_k	l_k	RX	...	a_k
...	...	...	...	...	...	...
n-1	0	S_{n-1}	l_{n-1}	RWA	...	-

- Segment length
- Protection bits: 프로세스가 해당 segment에 대해 갖고 있는 권한

Segmentation System: Address Mapping



1. **프로세스의 SMT가 저장되어 있는 주소 b에 접근**
2. **SMT에서 segment s의 entry 찾음**
   - s의 entry 위치 = b + s * entrySize
3. **찾아진 Entry에 대해 다음 단계들을 순차적으로 실행**
   - ① 존재 비트가 0 인 경우, (segment fault)
     swap deive로부터 해당 segment를 메모리로 적재
     SMT를 갱신
   - ② 변위(d)가 segment 길이보다 큰 경우 ($d > l_s$),
     segment overflow exception 처리 모듈을 호출
   - ③ 허가되지 않은 연산일 경우 (protection bit field 검사),
     segment protection exception 처리 모듈을 호출
4. **실제 주소 r 계산 ($r = a_s + d$)**
5. **r로 메모리에 접근**

- Segment Fault: cf. page fault
- Segment Overflow
  - displacement d가 segment 길이보다 큰 경우
- Segment Protection
  - 프로세스 권한 확인

Memory Management

partition	start address	size	current process ID	segment number	storage protection key	other fields
0	...	...	Px	x1	...	...
1	...	...	none	...	...	...
2	...	...	Py	y1	...	...
4	...	...	Px	x2	...	...
5	...	...	none	...	...	...
6	...	...	...	...	...	...

- VPM과 유사
  - Segment 적재 시, 크기에 맞춰 분할 후 적재

Segment Sharing & Protection

- 논리적으로 분할되어 있어, 공유 및 보호 용이
  - 이전에는 논리적으로 분할되지 않아 문제 발생했음
  - 이제 length + start address로 같은 segment 내에서 이동하면 됨

Segmentation System: Summary

- 정의
  - 프로그램을 논리 단위로 분할(Segment)
  - 메모리를 동적으로 분할
- 특징
  - 내부 단편화 발생 X
  - Segment sharing/protection 용이
  - Paging System 대비 관리 overhead 큼
- 필요한 segment만 메모리에 적재해 사용 ⇒ 메모리의 효율적 활용
- Segment Mapping Overhead
  - Segment 매핑을 위해 메모리 공간 및 추가 메모리 접근 필요
  - 전용 HW 활용으로 해결 가능