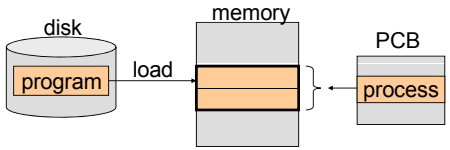
메모리 관리

연세대학교

배경 지식

연세대학교

- 프로그램은 주기억장치(메모리)에 적재되어야 실행될 수 있다.

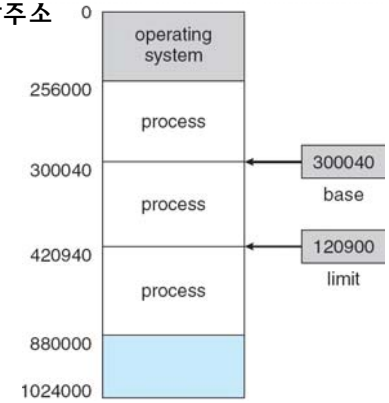


- 다중 프로세스 실행 환경에서 프로그램이 올바르게 동작하기 위해서는 메모리 보호가 필요하다
  - 다른 프로세스의 프로그램 메모리 접근을 방지해야 함

메모리 보호 방법

연세대학교

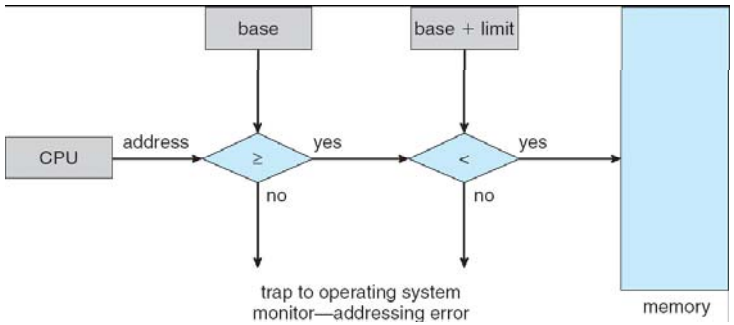
- 각 프로세스에게 분리된 메모리 공간을 제공함
- base 와 limit 레지스터 사용하여 유효 메모리 주소 범위 지정
  - base 레지스터: 메모리 공간 시작주소
  - limit 레지스터: 메모리 공간 크기
  - 유효한 메모리 주소 범위:
    base ≤ address < base+limit



메모리 보호 방법(2)

연세대학교

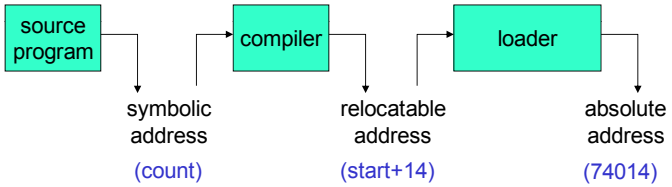
- 메모리 주소 검사 과정



주소 바인딩(Address Binding)

연세대학교

- 프로그램을 작성할 때에 프로그래머는 기호주소(Symbolic Address)를 사용한다.
- 주소 바인딩
  - 프로그램이 실행되기 전에 기호 주소를 메모리 주소로 바인딩(binding)해주어야 한다.



주소 바인딩 시기

연세대학교

- 컴파일 시간 바인딩
  - 프로그램의 적재 위치를 사전에 알고 있으면 컴파일을 할 때에 주소를 바인딩 한다.
- 적재 시간 바인딩
  - 프로그램의 적재 위치를 사전에 모르면 컴파일러는 재배치 가능 코드(relocatable code)를 생성한다.
  - 프로그램이 메모리에 적재될 때에 실제 주소를 바인딩 한다.
- 실행 시간 바인딩
  - 프로그램 실행 도중에 프로세스의 적재 위치가 바뀔 수 있는 경우에는 실행 동안 주소 바인딩이 이루어진다.
  - 컴파일러는 논리 주소를 사용한 코드를 생성하며 실행시간 동안 MMU(메모리 관리 장치)에 의해서 논리 주소가 실제 메모리주소로 변환된다
  - 가장 많이 사용하는 방법이다.

논리 주소와 물리적 주소

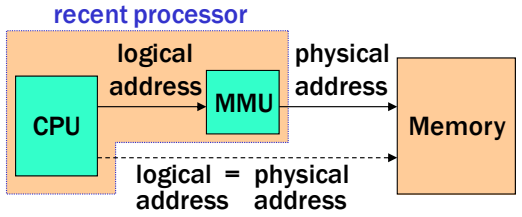
연세대학교

- 논리 주소와 물리적 주소
  - **논리 주소**(logical address): 프로그램이 사용하는 주소
  - **물리적 주소**(physical address): 실제 메모리 장치의 주소
- 컴파일 시간 바인딩과 적재 시간 바인딩에서
  - 논리 주소와 물리적 주소는 같다.
- 실행 시간 바인딩에서
  - 논리 주소와 물리적 주소는 같지 않다.
  - 논리 주소를 **가상 주소**(virtual address)라고도 한다.
  - 프로그램은 물리적 주소에 대해서 알 수 없다.
    (고급 언어에서 다루는 포인터 값은 논리 주소이다.)

논리 주소와 물리적 주소(2)

연세대학교

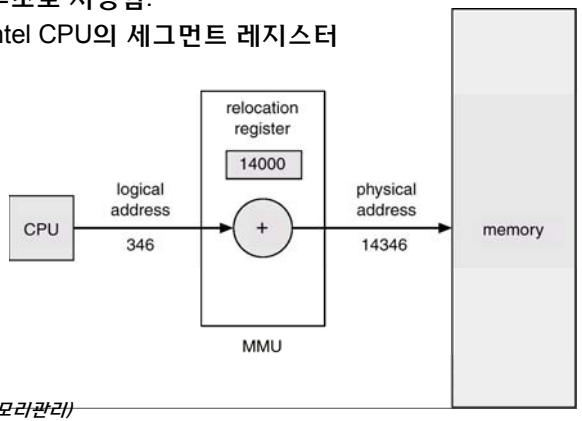
- MMU가 없는 CPU에서는
  - 논리 주소와 물리적 주소가 같다.
  - 간단한 CPU에는 MMU가 없다.
- MMU를 포함한 CPU에서는
  - 논리 주소와 물리적 주소가 다르다.
  - 최근의 많은 CPU는 주소변환을 담당하는 MMU를 포함하고 있다.



동적 재배치

연세대학교

- 가장 간단한 주소 변환 방법은 재배치(relocation) 레지스터를 사용하는 방법이다.
  - 운영체제가 재배치 레지스터 값을 프로그램이 적재된 메모리의 시작주소로 지정함.
  - (예) Intel CPU의 세그먼트 레지스터



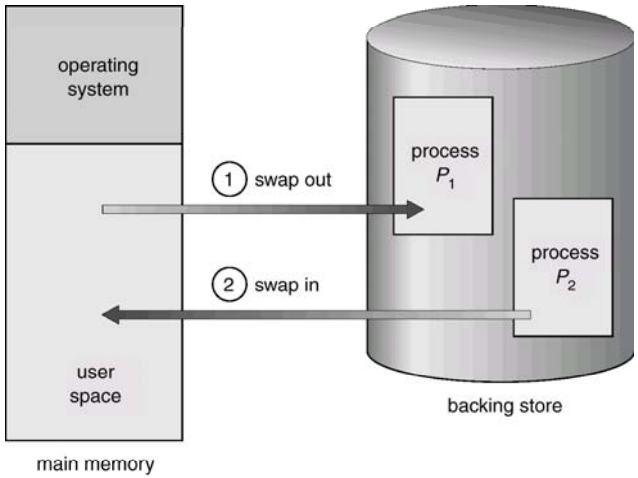
스와핑(Swapping)

연세대학교

- 스와핑
  - 생성된 프로세스가 많은 경우에 모든 프로세스를 메모리에 적재할 수 없다.
  - 이 경우에 어떤 프로세스가 일시적으로 메모리에서 디스크로 옮겨진 후에(swap out), 나중에 다시 메모리에 적재되어(swap in) 실행을 재개할 수 있다. 이렇게 하여 확보된 메모리 공간은 다른 프로세스에게 할당된다. → 이 과정을 스와핑이라고 한다.
- 스와핑과 주소바인딩
  - 컴파일시간 또는 적재시간 바인딩을 하는 경우에는 같은 주소에 적재되어야 한다.
  - 실행시간 바인딩을 하는 경우에는 다른 주소에 적재될 수 있다.

스와핑 (2)

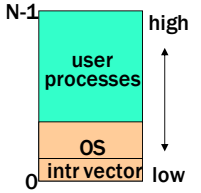
연세대학교

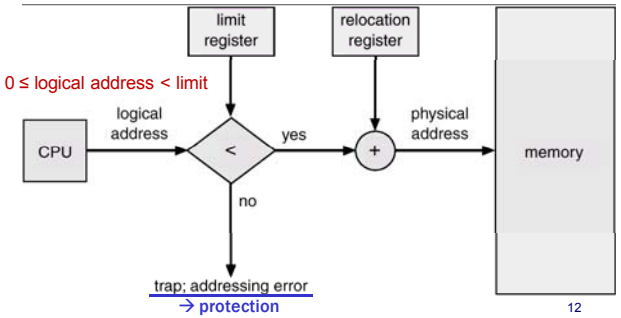


연속적인 메모리 공간 할당

연세대학교

- 메모리 공간의 분할 사용
  - 운영체제 (메모리 상주 부분)
  - 사용자 프로세스들
- 연속적인 메모리 공간 할당
  - 각 프로세스는 메모리의 연속된 단일 영역을 할당 받음
- 재배치 레지스터를 사용한 메모리 보호

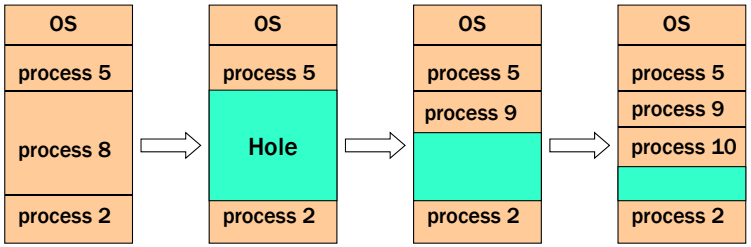




메모리 할당

연세대학교

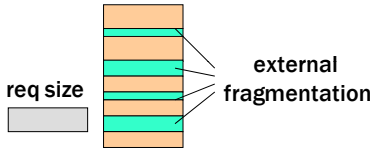
- 홀(hole) – 사용되지 않는 메모리 영역들
- 운영체제는 프로세스에 할당된 영역과 사용되지 않은 영역을 관리함
- 프로세스가 메모리 공간을 요청하면
  - 운영체제는 요청을 수용할 수 있는 크기의 홀을 찾아서
  - 이 홀에서 요청한 메모리 공간을 프로세스에게 할당함



외부단편화와 압축

연세대학교

- 외부 단편화(external fragmentation)
  - 메모리 할당이 반복되면 작은 크기의 홀들만 존재하게 될 수 있다
  - 이 경우에 모든 홀의 크기의 합은 요청한 크기보다 크지만 연속된 공간이 아니어서 할당할 수 없는 경우에 발생한다.



- 해결책
  - 압축(compaction)
    - 현재 할당되어 있는 프로세스의 적재 위치를 재배치하여 하나의 큰 홀을 만든다.
  - 비연속적인 메모리 공간 할당 – 페이징, 세그먼테이션

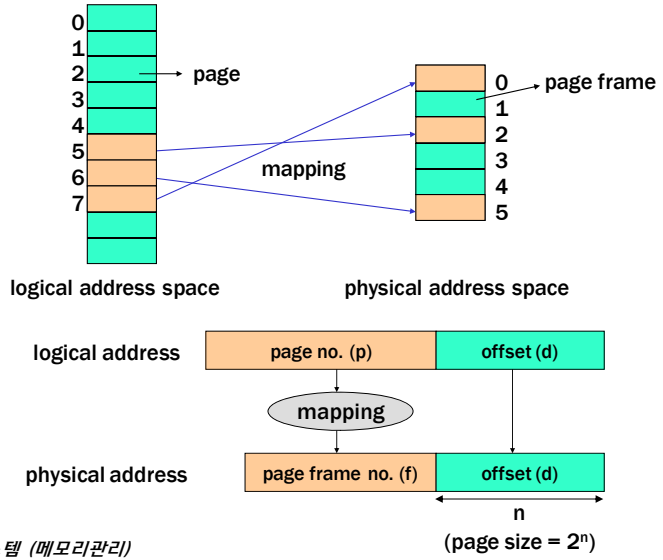
페이징 (Paging)

연세대학교

- 프로세스에게 연속적이 아닌 메모리 공간 할당을 허용함
- 페이지와 페이지 프레임
  - 프로세스의 논리적인 주소공간을 일정 크기의 page로 나눔
  - 물리적 메모리를 같은 크기의 page frame으로 나눔
  - 페이지 크기는 대개 512B ~ 8KB 임 (80386: 4KB)
- 페이징
  - 물리적 메모리의 할당은 page 단위로 이루어짐
  - 프로세스의 각 page에 물리적 메모리의 사용 중이 아닌 page frame을 할당함
- 페이징에서의 주소 변환
  - 주소변환 하드웨어를 사용하여 논리적 주소를 물리적 주소로 변환함
  - 페이지 번호를 페이지 프레임 번호로 변환하고 페이지 내의 주소는 그대로 사용함

페이징에서의 주소 변환

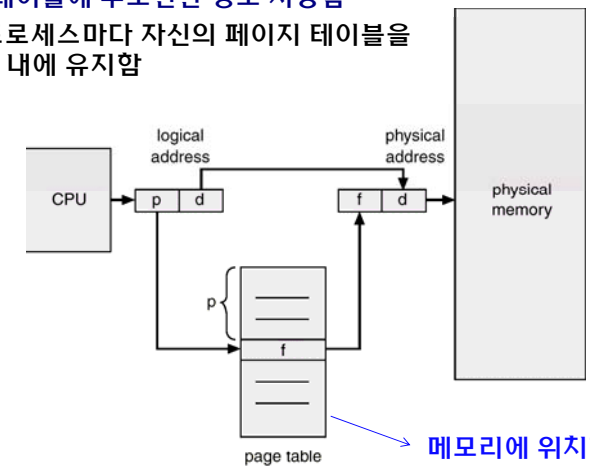
연세대학교



페이징 하드웨어

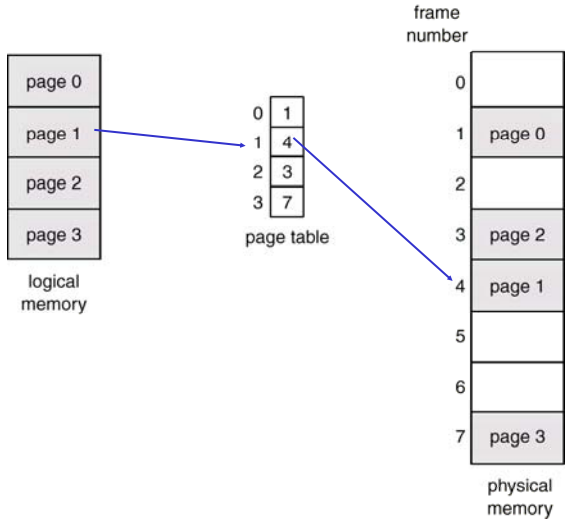
연세대학교

- 페이지 테이블에 주소변환 정보 저장함
  - 각 프로세스마다 자신의 페이지 테이블을 PCB 내에 유지함



페이징의 예

연세대학교



페이지 할당

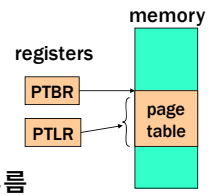
연세대학교

- 페이지 할당
  - 운영체제는 사용 중이 아닌 페이지 프레임(free frame)을 관리함
  - 프로세스가 요청한 페이지 개수만큼 free frame을 할당함
  - 할당된 내용을 반영하도록 페이지 테이블을 설정함
- 내부 단편화(internal fragmentation) 문제
  - 페이징은 외부 단편화가 발생하지 않음
  - 대신에 페이지 내부에 사용하지 않는 부분이 존재할 수 있음
  - 각 프로세스의 내부 단편화 크기의 평균은 ½ 페이지 크기임
  - 페이지 크기를 적절하게 정하는 것이 중요함
    - 작은 페이지 크기: 작은 내부단편화, 큰 페이지 테이블
    - 큰 페이지 크기: 큰 내부단편화, 작은 페이지 테이블, 효율적 페이지 입출력

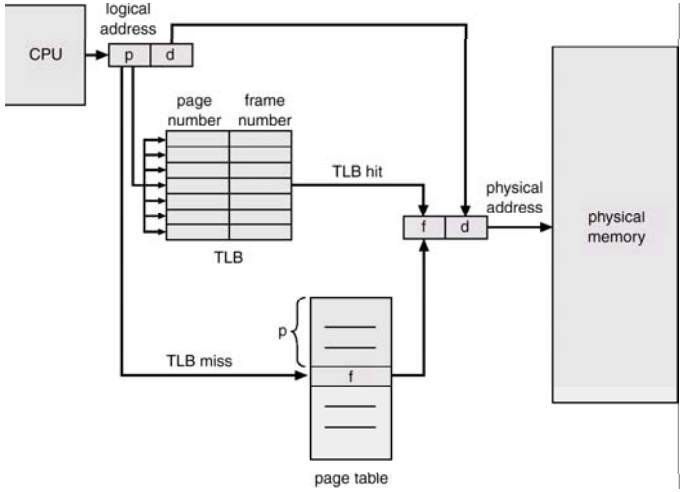
페이지 테이블과 TLB

연세대학교

- 페이지 테이블
  - 대개 메모리의 PCB내에 위치함
  - 프로세스가 실행될 때에 PTBR(page table base register)에 페이지 테이블의 시작주소를, PTLR(page table length register)에 페이지 테이블의 크기를 저장하여 페이지 테이블을 참조함
- 페이징 방법의 문제점
  - 메모리 접근을 위해서 페이지 테이블의 접근이 필요함 → 두 번의 메모리 액세스
- 해결책: TLB 사용
  - 페이지 테이블용 캐시를 사용함 → Translation Look-aside buffer (TLB)라고 부름
  - 대개 완전연관 맵핑을 사용함
- 프로세스가 바뀔 때마다(context 전환) TLB내용은 지워진다.



TLB를 사용한 페이징 하드웨어



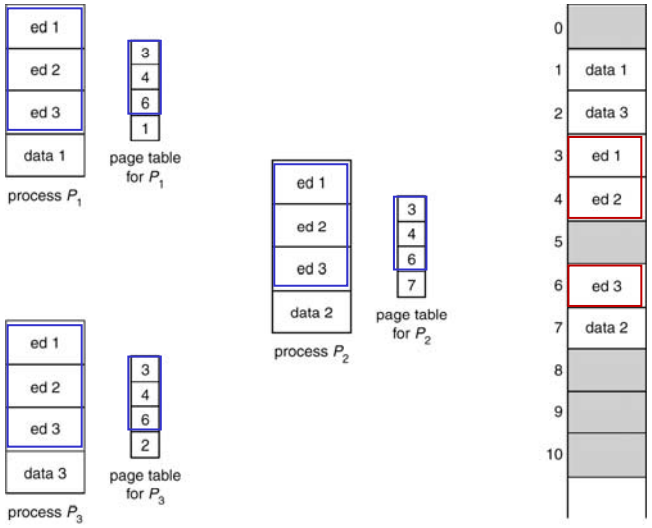
페이징에서의 메모리 보호

- 페이징 기법에서 메모리 보호는 페이지 단위로 이루어진다.
  - 페이지 테이블에 페이지 프레임 번호 뿐 만 아니라 메모리 보호를 위한 추가 정보 기록
- 보호 비트 (P)
  - 페이지에 대한 사용 권한을 지정
  - 읽기전용, 읽기쓰기가능, 실행전용 등의 권한 정보를 기록
- 유효 비트 (V)
  - 페이지 테이블 항목의 유효 여부 표시

	frame no.	P	V
0		eo	1
1		eo	1
2		rw	1
3		ro	1
4		-	0

page table

페이징에서의 메모리 공유



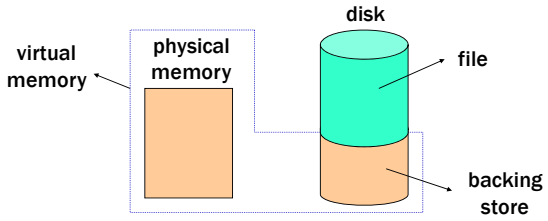
가상메모리의 배경 지식

- 프로그램을 실행하는 동안 프로그램의 일부분만 사용할 수 있다.
  - 오류 처리 루틴은 오류가 발생할 때에만 실행된다.
  - 어떤 루틴은 해당 옵션이 선택되었을 때에만 실행된다.
  - 배열, 테이블이 필요한 크기보다 큰 공간을 할당 받는다.
- 프로그램을 실행하는 동안 프로그램 전체를 사용하는 경우에도 어떤 순간에는 프로그램 전체를 필요로 하지 않는다.
  - 프로그램의 어떤 부분은 사용하지 않을 수도 있다.
- 따라서 프로그램의 모든 부분이 메모리에 적재될 필요는 없다.
  - 프로그램 실행에 필요한 부분만 메모리에 적재하고
  - 나머지 부분은 보조기억장치에 저장한다.

가상 메모리

연세대학교

- 가상 메모리
  - 가상 메모리는 물리적 메모리와 보조기억장치 공간으로 구성
  - 프로그램의 논리적인 주소공간을 가상 메모리 공간에 맵핑함
    - 지금 프로그램 실행에 필요한 부분은 물리적 메모리에 적재
    - 나머지 부분은 물리적 메모리 또는 보조기억장치에 저장



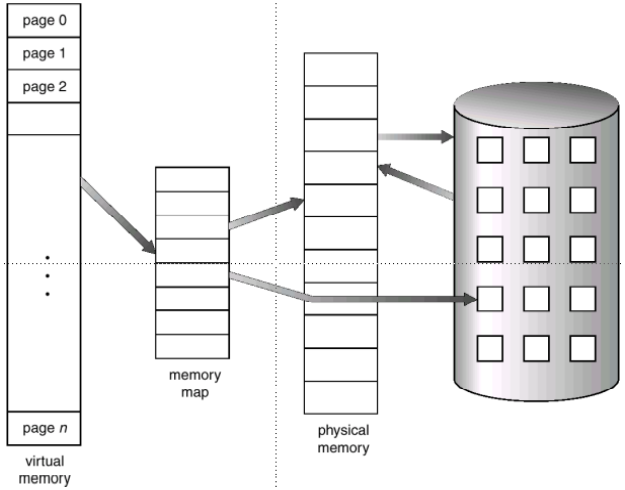
가상 메모리의 장점

연세대학교

- 가상 메모리의 장점
  - 논리적 메모리(주소공간)과 물리적 메모리를 분리할 수 있다.
  - 물리적 메모리보다 큰 가상 메모리를 제공한다.
  - 물리적 메모리 크기보다 큰 프로그램을 실행시킬 수 있다.
  - 메모리에 완전히 적재되지 않은 프로세스도 실행시킬 수 있다.
  - 동시에 메모리에 적재시킬 수 있는 프로그램 수가 증가한다.
  - 페이지 단위로 스와핑을 수행할 수 있다. → 스와핑 시간 단축

가상 메모리와 물리적 메모리

연세대학교

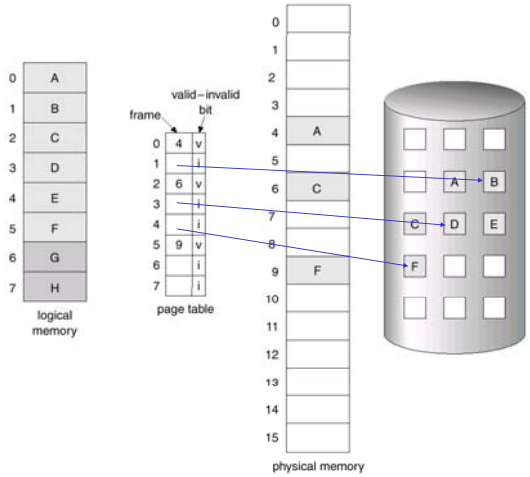


요구 페이징 (Demand Paging)

연세대학교

- 가상 메모리는 요구 페이징 기법을 사용하여 구현된다.
- 요구 페이징
  - 페이지가 필요할 경우에만 메모리에 적재함
- 요구 페이징 구현
  - 페이지 테이블의 유효비트를 이용하여 페이지의 맵핑 위치 구분
    - 유효비트=1: 메모리에 적재된 페이지
    - 유효비트=0: 디스크에 있거나 유효하지 않은 페이지
  - 페이지를 참조할 때에
    - 유효비트=1이면 페이징 주소변환을 거쳐 메모리 액세스
    - 유효비트=0이면 page fault 트랩 발생
      - 디스크에 있는 경우에 빈 페이지 프레임 공간을 확보한 후에 디스크 입출력을 이용하여 참조 페이지를 메모리에 적재한 후에 다시 페이지 참조
      - 유효하지 않은 페이지의 경우 프로세스 종료

메모리에 적재되지 않은 페이지



Page Fault 처리 과정