Chapter 7.
Memory Management

7장의 강의 목표

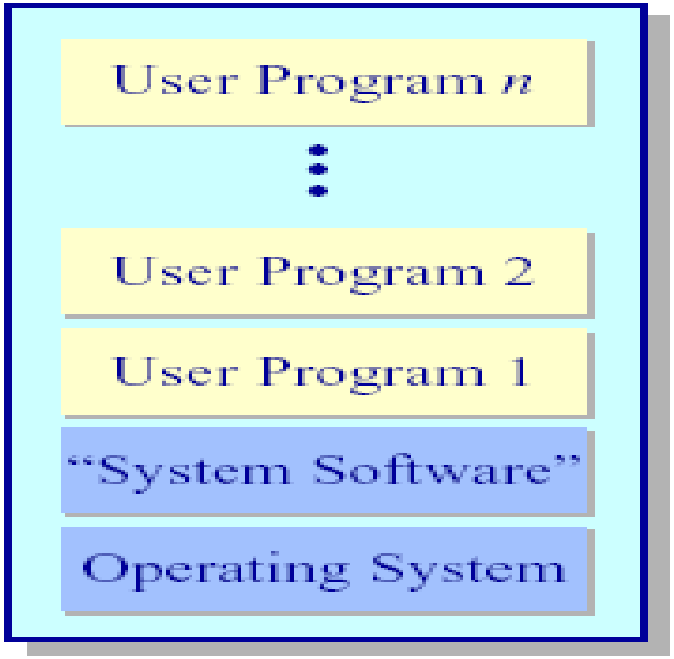
- 메모리 관리 기법의 기본적인 요구조건을 이해한다.
- 메모리 분할 기법을 알아본다.
- 페이징과 세그먼테이션 기법을 이해한다.
- 로딩과 링킹에 대해서 알아본다.

목 차

7.1 메모리 관리 요구조건
7.2 메모리 분할 (memory partitioning)
7.3 페이징 (paging)
7.4 세그먼테이션 (segmentation)
부록: 로딩과 링킹

7.1 Memory Management Requirements

- Uniprogramming system
  - ✓ main memory is divided into two parts
    - Kernel part: operating system
    - User part: program currently being executed
- Multiprogramming system
  - ✓ User part is further subdivided into multiple programs
  - ✓ Some portions of program are resident in secondary storage



메모리 관리

- 메모리 관리 (memory management)
  - ✓ 다중 프로그래밍 시스템에서 다수의 프로세스를 수용하기 위해 주기억장치를 동적으로 분할하는 작업

- 메모리 관리 요구조건
  - ✓ 재배치(Relocation)
  - ✓ 보호(Protection)
  - ✓ 공유(Sharing)
  - ✓ 논리적 구성(Logical organization)
  - ✓ 물리적 구성(Physical organization)

메모리 관리 요구조건

- 재배치(Relocation)
  - ✓ 다수의 프로세스들이 스왑인(swap in), 스왑아웃(swap out) 시 다른 주소 공간으로의 프로세스 재배치 필요
  - ✓ 재배치 고려한 프로세스 주소 지정 요구조건

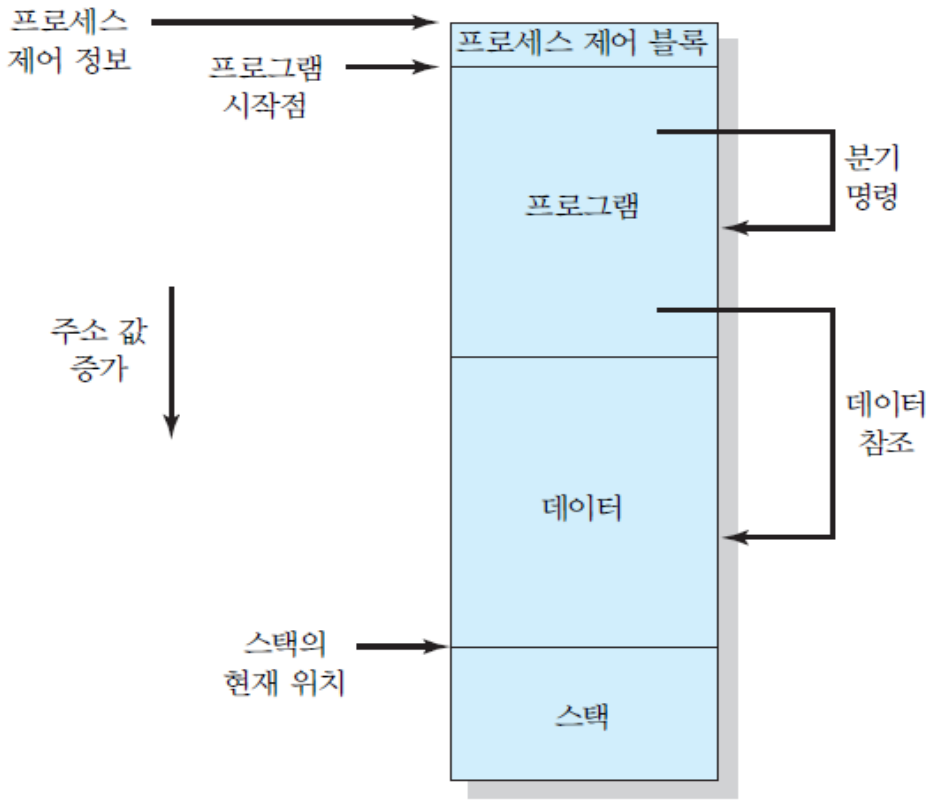


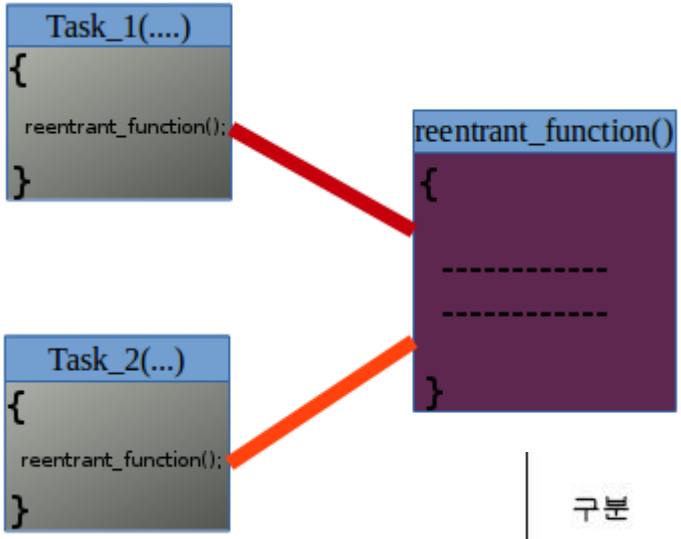
그림 7.1 프로세스의 주소 지정 요구 조건

메모리 관리 요구조건

- 보호(Protection)
  - ✓ 다른 프로세스들의 간섭으로부터 보호
  - ✓ 메모리 참조 검사: 실행 중 해당 프로세스에 할당된 메모리 공간만 참조되었는지 확인 필요
  - ✓ 메모리 보호 : 처리기(하드웨어)적인 검사 요구
    - OS에서 프로그램이 만들어내는 모든 메모리 참조를 예측할 수도 없고, 가능하더라도 오버헤드 심함

메모리 관리 요구조건

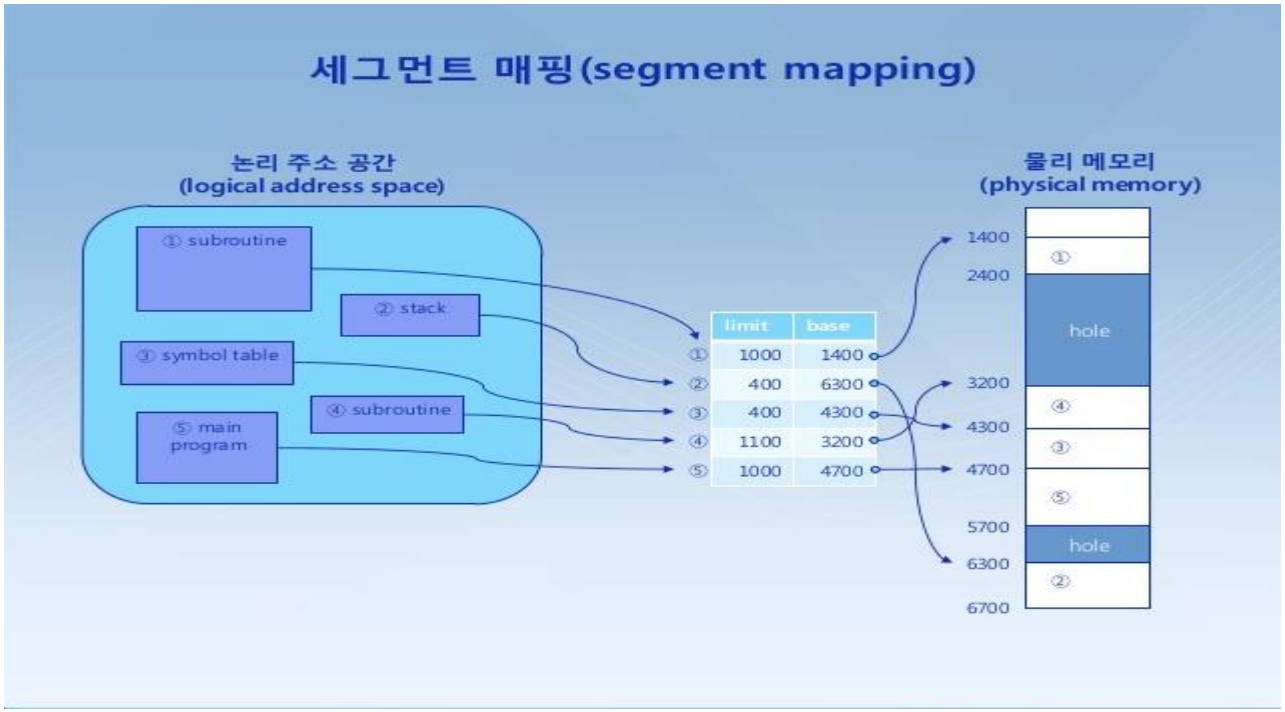
- 공유(Sharing)
  - ✓ 주기억장치의 같은 부분을 접근하려는 여러 개의 프로세스들을 융통성 있게 허용
    - Reentrant-code
  - ✓ 필수적인 보호 기능을 침해하지 않는 범위에서 제한된 접근을 통한 공유



구분	Reentrant Code	일반프로그램
용도	공통 기능을 동시에 재사용이 가능함	동시에 재사용 불가
변수의 유일성	Local 변수로 유일성 보장	보장 않됨
메모리 영역	재진입시 별도 영역을 설정하고 퇴출시 해제됨	프로그램 종료시 해제됨
특징	상호배제구현,환경정보 저장/해제등 로직이 복잡함	로직이 간단하나, 멀티프로그램, 멀티사용자 구현 불가

메모리 관리 요구조건

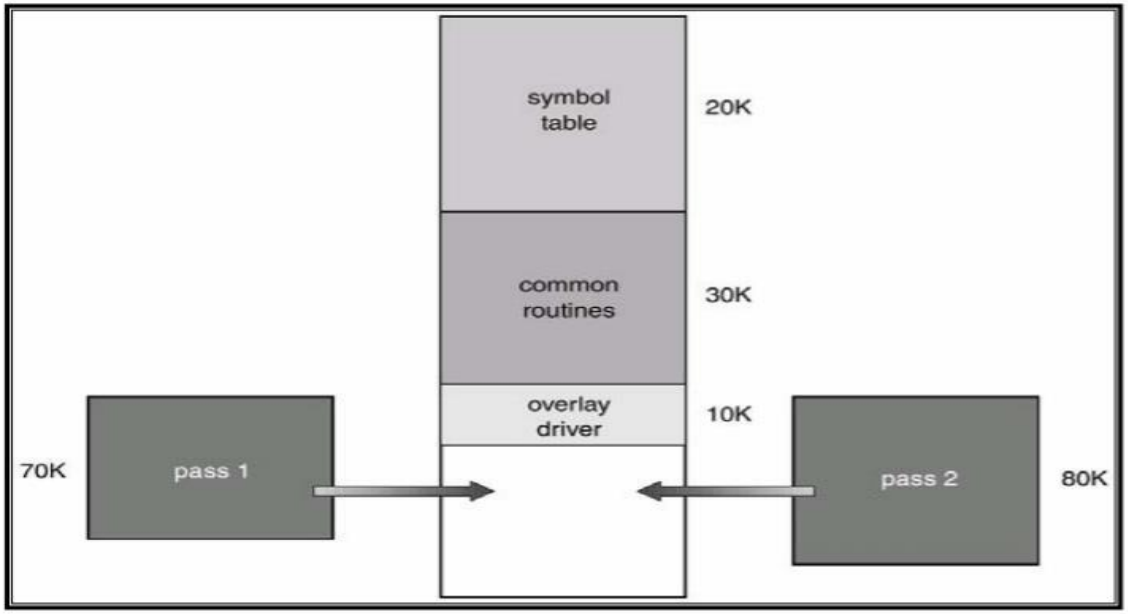
- 논리적인 구성
  - ✓ 일반적인 프로그램 : 모듈 단위 구성
  - ✓ 운영체제 및 하드웨어의 모듈 단위 처리 시 이점
    - 모듈의 작성과 컴파일 독립적으로 이루어짐
    - 비교적 적은 추가비용(overhead)으로 모듈마다 서로 다른 보호 등급(r-w-x) 적용가능
    - 프로세스 간 모듈 공유 기법 제공
  - ✓ 대표적인 메모리 관리 기술 : 세그먼테이션(Segmentation)



메모리 관리 요구조건

- 물리적인 구성
  - ✓ 주기억 장치와 보조 기억 장치 사이의 정보 흐름 구성
  - ✓ 정보 흐름 구성 책임자 : 시스템
    - "사용가능한 주기억장치 용량 < 프로그램 및 데이터" 인 경우 처리
      - Overlay(중첩) 기법 이용
    - 멀티 프로그래밍 환경 : 사용가능한 공간의 양과 위치 정보를 프로그래머는 알 수 없음.

Overlays for a Two-Pass Assembler



메모리 관리 기법

- 연속 메모리 관리(contiguous memory management)
  - ✓ 프로그램 전체가 하나의 커다란 공간에 연속적으로 할당되어야만 함
    - 단일 연속 메모리 관리
    - 고정 분할 기법 (fixed partitioning)
    - 동적 분할 기법 (dynamic partitioning)

- 불연속 메모리 관리(non-contiguous memory management)
  - ✓ 프로그램의 일부가 서로 다른 주소 공간에 할당될 수 있는 기법
    - 고정크기: 페이징 (paging)
    - 가변크기: 세그먼테이션 (segmentation)

7.2 Memory Partitioning

- Fixed partitioning
  - ✓ Partition Sizes
  - ✓ Internal fragmentation
  - ✓ Placement Algorithm
- Dynamic partitioning
  - ✓ External Fragmentation
  - ✓ Compaction
  - ✓ Placement Algorithm
- Buddy system
  - ✓ Tradeoff between Fixed and Dynamic partitioning
- Relocation
  - ✓ Logical address (relative address)
  - ✓ Physical address (absolute address)

메모리 관리 기법

- 메모리 관리 기법(1/3)

기술	설명	강점	약점
고정 분할 (fixed partitioning)	시스템 생성 시에 주기억장치가 고정된 파티션들로 분할된다. 프로세스는 균등사이즈의 파티션 또는 그보다 큰 파티션으로 적재된다.	구현이 간단하다 :운영체제에 오버헤드가 거의 없다	내부단편화로 인한 비효율적인 사용; 최대 활성 프로세스의 수가 고정됨.
동적 분할 (dynamic partitioning)	파티션들이 동적으로 생성되며, 각 프로세스는 자신의 크기와 일치하는 크기의 파티션에 적재된다.	내부단편화가 없고 주기억장치를 보다 효율적으로 사용할 수 있다.	외부 단편화를 해결하기위한 메모리 집약(compaction)이 요구된다. 따라서 처리기 효율이 나빠진다.

메모리 관리 기법

메모리 관리 기법(2/3)

기술	설명	강점	약점
단순 페이징 (simple paging)	주기억장치는 균등사이즈의 프레임으로 나뉜다. 각 프로세스는 프레임들과 같은 길이를 가진 균등페이지들로 나뉜다. 프로세스의 모든 페이지가 적재되어야 하며 이 페이지를 저장하는 프레임들은 연속적일 필요는 없다.	외부 단편화가 없다	적은양의 내부 단편화가 생긴다
단순 세그먼테이션 (simple segmentation)	각 프로세스는 여러 세그먼트들로 나뉜다. 프로세스의 모든 세그먼트가 적재되어야 하며 이 세그먼트를 저장하는 동적 파티션들은 연속적일 필요는 없다.	내부단편화가 없고 메모리 사용 효율이 개선되며, 그리고 동적분할에 비해서 오버헤드가 적다	외부 단편화

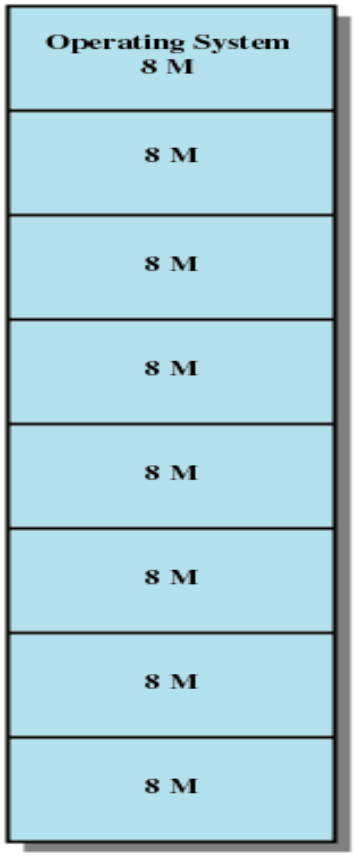
메모리 관리 기법

메모리 관리 기법(3/3)

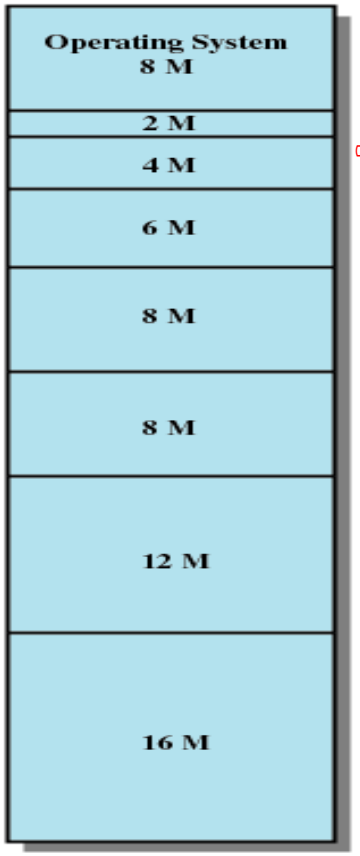
기술	설명	강점	약점
가상 메모리 페이징 (virtual memory paging)	단순 페이징과 비교해서 프로세스의 페이지 전부를 로드시킬 필요가 없다. 필요한 페이지가 있으면 후에 자동적으로 불려들어진다.	외부 단편화가 없다. 다중 프로그래밍 정도가 높으며, 가상 주소 공간이 크다	복잡한 메모리관리의 오버헤드
가상 메모리 세그먼테이션 (virtual memory segmentation)	단순 세그먼테이션과 비교해서 필요하지 않은 세그먼트들을 로드하지 않는다. 필요한 세그먼트가 있으면 나중에 자동적으로 불려들어진다.	내부단편화가 없다. 높은 수준의 다중 프로그래밍, 큰 가상 주소공간, 보호와 공유를 지원	복잡한 메모리관리의 오버헤드

Memory Partitioning

- Fixed partitioning: partition size



(a) Equal-size partitions



(b) Unequal-size partitions

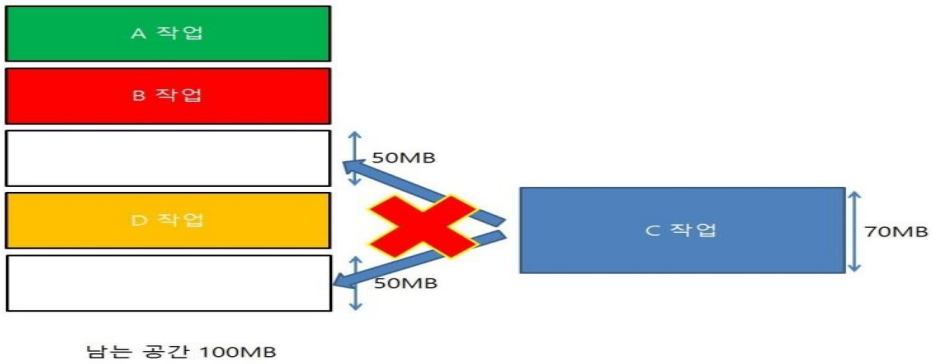
☞ internal fragmentation 줄일수 있음

☞ overlay 없이 로드가능

Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory

메모리 분할: fixed partitioning

- 고정 분할(1/2)
  - ✓ 균등 분할의 문제점
    - 프로그램 > 파티션 가능성: overlay 기법 이용해야 함
    - 주기억 장치 이용률 저조
    - 내부단편화 발생(internal fragmentation)
      - 적재되는 데이터가 파티션보다 작을때 파티션 내부 공간의 낭비가 발생하는 현상

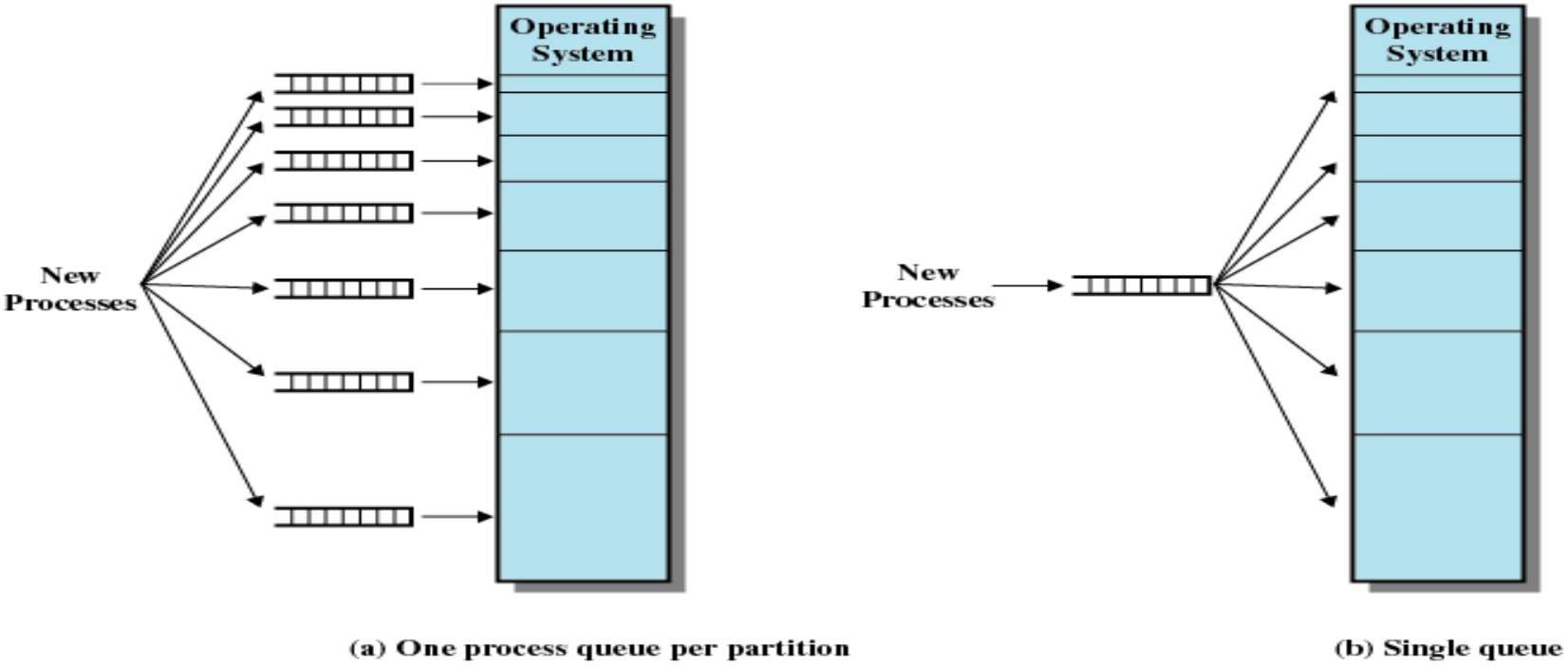


  - ✓ 배치 알고리즘
    - 균등 분할 : 사용가능한 파티션에 적재 or
      모든 파티션이 사용중이면 기존 프로세스 스왑 아웃
    - 비균등 분할 : 최적 파티션에 할당(내부 단편화 최소화)
      파티션 당 하나의 프로세스 큐 방식 vs 단일 큐 방식(그림 7.3)

Memory Partitioning

고정 분할(2/2)

- ✓ 고정 분할 기법의 문제점
  - 파티션 수에 의해 활성화된 프로세스 수 제한
  - 크기 작은 작업일수록 파티션 공간 비효율적 사용
  - 내부 단편화 발생



(a) One process queue per partition

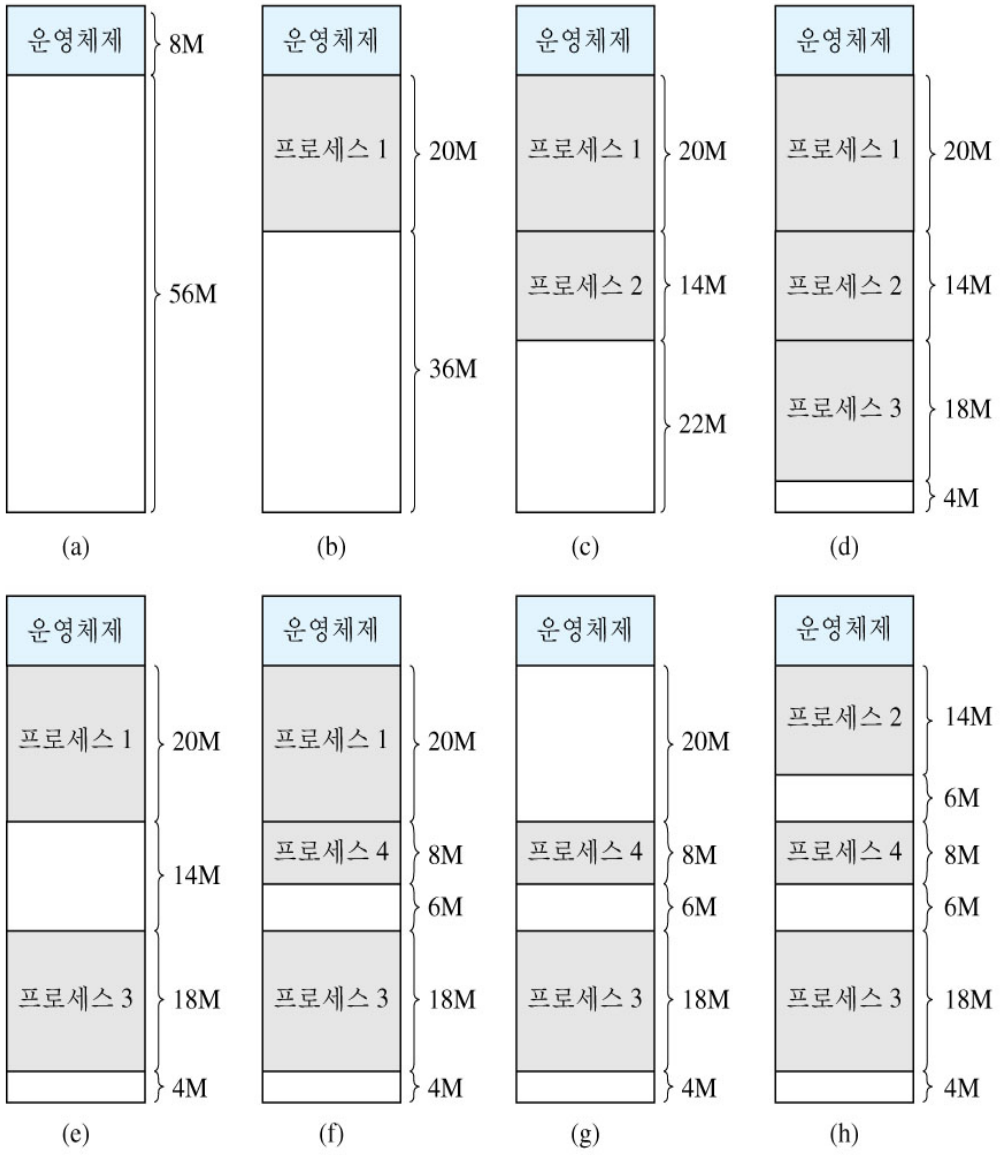
(b) Single queue

Figure 7.3 Memory Assignment for Fixed Partitioning

☞ absolute vs relocatable loading

메모리 분할: dynamic partitioning

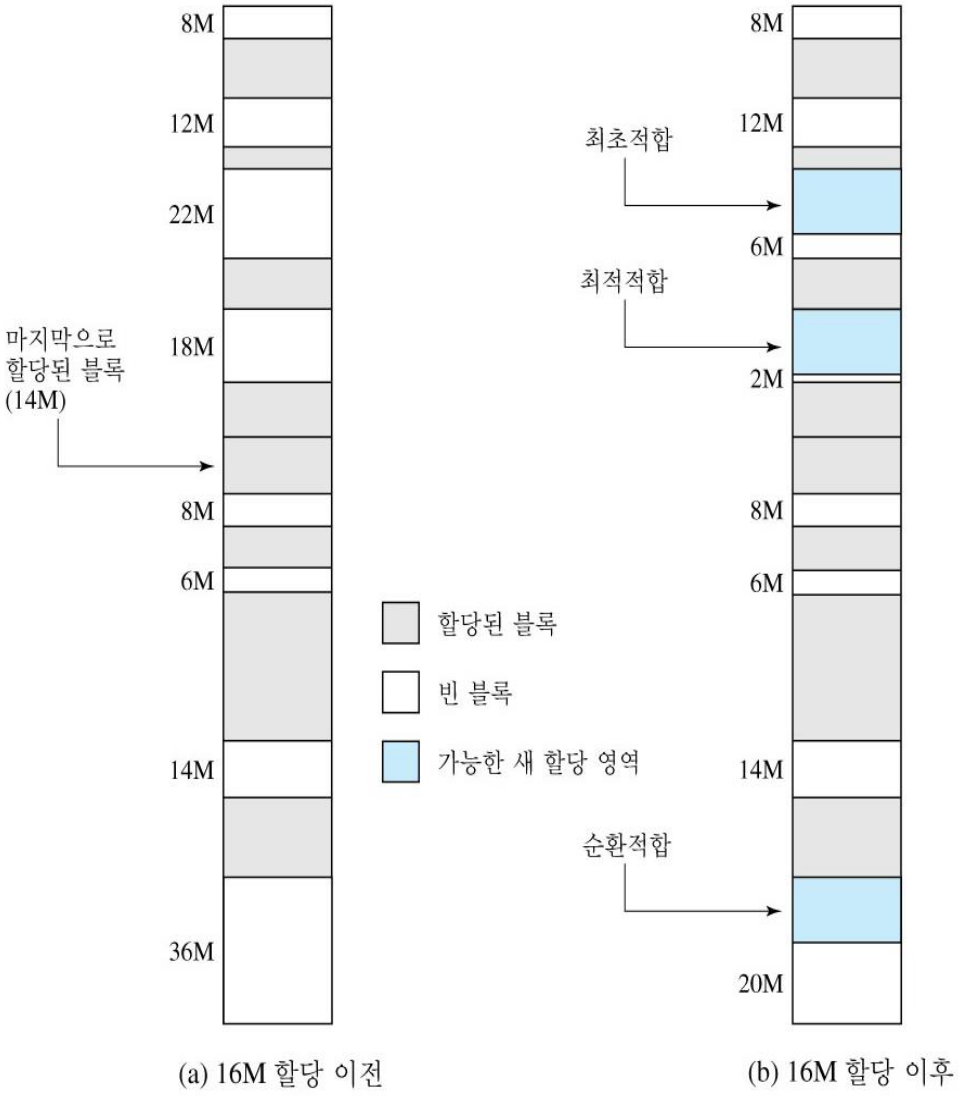
- 동적 분할(1/2)
  - ✓ 파티션 크기와 개수가 가변적
  - ✓ 프로세스가 요구한 크기만큼의 메모리만 할당
  - ✓ 외부 단편화 발생
    - Hole에 의해 시간이 지날수록, 메모리 단편화는 심해지고 메모리 이용률감소
    - 모든 파티션 영역이외의 메모리가 점차 사용할 수 없는 조각으로 변하는 현상
  - ✓ 메모리 집약(compaction) 필요
    - Windows 조각모음 기능



<그림 7.4> 동적 분할의 결과

메모리 분할: dynamic partitioning

- 동적 분할(2/2)
  - ✓ 배치 알고리즘
    - 최적적합 (best-fit)
      - 가장 성능이 나쁨
    - 최초적합 (first-fit)
      - 가장 간단하고 대부분의 경우 최적이며, 가장 빠름
    - 순환적합 (next-fit)
  - ✓ 교체 알고리즘
    - 교체될 프로세스의 선택



<그림 7.5> 16Mbyte 블록의 배정 전과 후의 메모리 구성의 예

Memory Partitioning

- Buddy system
  - ✓ Motivation
    - drawbacks of fixed partitioning schemes: limit the number of active processes, inefficient use of memory space
    - drawbacks of dynamic partitioning schemes: complex to maintain, overhead of compaction
    - (절충안)Compromise between efficiency and simple management
    - 그러나, 현재 OS에서는 페이징과 세그멘테이션 기반 가상 메모리 기법을 주로 사용하며, 버디시스템은 병렬 프로그램 할당과 해제 수단으로 사용
  - ✓ Block allocation of Buddy system
    - 2^L = 할당된 가장 작은 크기의 블록
    - 2^U = 할당된 가장 큰 크기의 블록, 보통 할당 가능한 전체 메모리의 크기와 같음.
    - Basic Idea
      - If a request of size *s* such that $2^{U-1} < s <= 2^U$, entire block is allocated
      - Otherwise block is split into two equal buddies(2^{U-1} 크기), until smallest block greater than or equal to *s* is generated

메모리 분할: buddy system

- 버디(Buddy) 시스템(1/3)
  - ✓ 고정 분할, 동적 분할 결점 보완한 절충안
  - ✓ 메모리 블록 크기
    - 2^k, $L \le K \le U$
    - 2^L = 할당된 가장 작은 크기의 블록
    - 2^U = 할당된 가장 큰 크기의 블록

```
void get_hole (int i)  {
    if( i==(U+1)) <실패>;
    if( <i_list empty> ){
        get_hole (i+1);
        <hole을 두 개의 버디로 나눈다.>
        <버디를 i_list에 포함시킨다.
    }
    <i_list의 첫 번째 hole을 선택한다.>
 }
```

< 동작 알고리즘 >

Memory Partitioning

- Buddy system: example
  - ✓ 할당 과정은 필요에 따라 나누어지고 합쳐지면서 진행됨

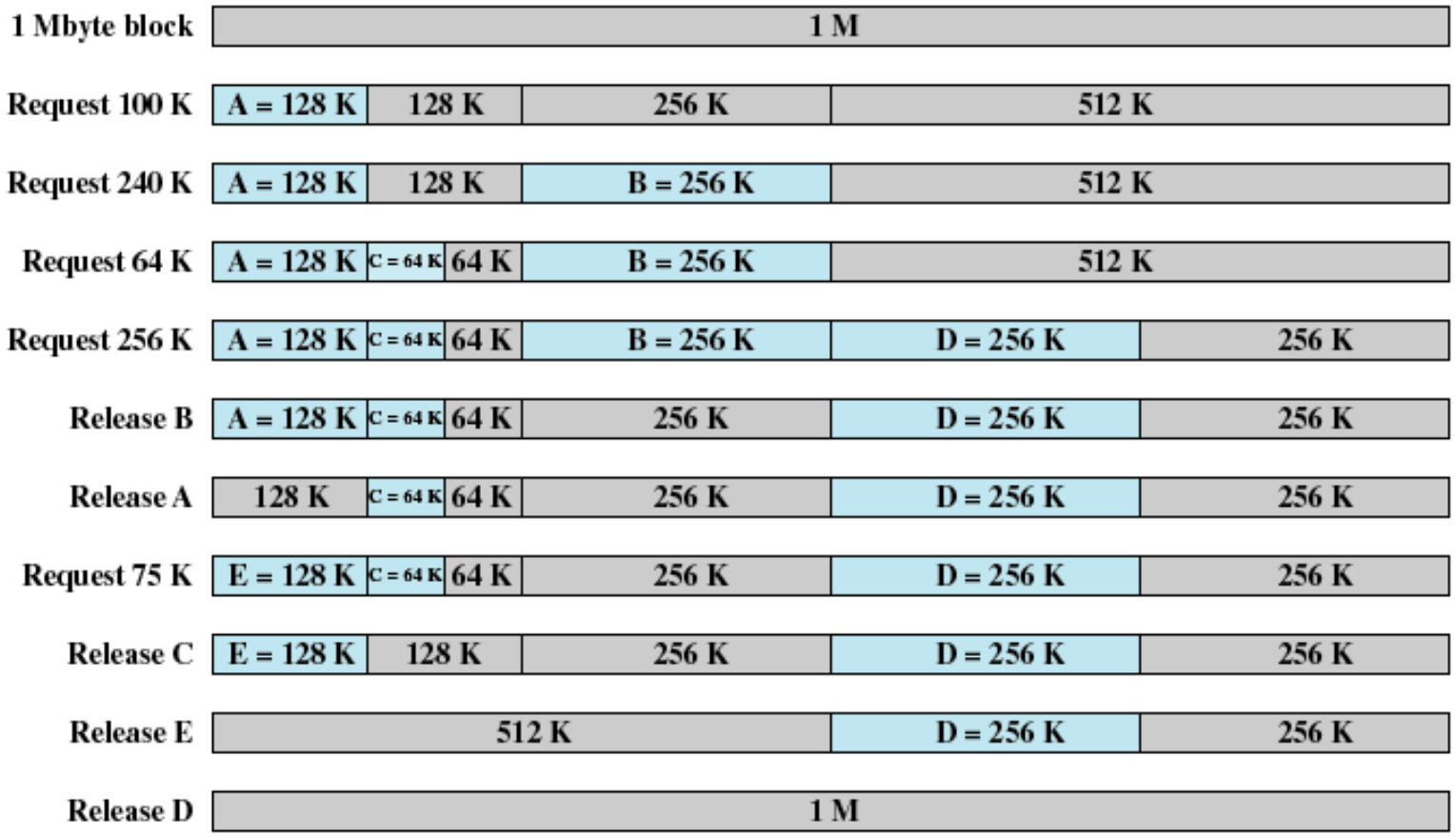


Figure 7.6 Example of Buddy System

Memory Partitioning

- Buddy system: representation
  - ✓ 이전 그림에서 “Releae B” 직후의 버디할당을 2진 트리로 표현

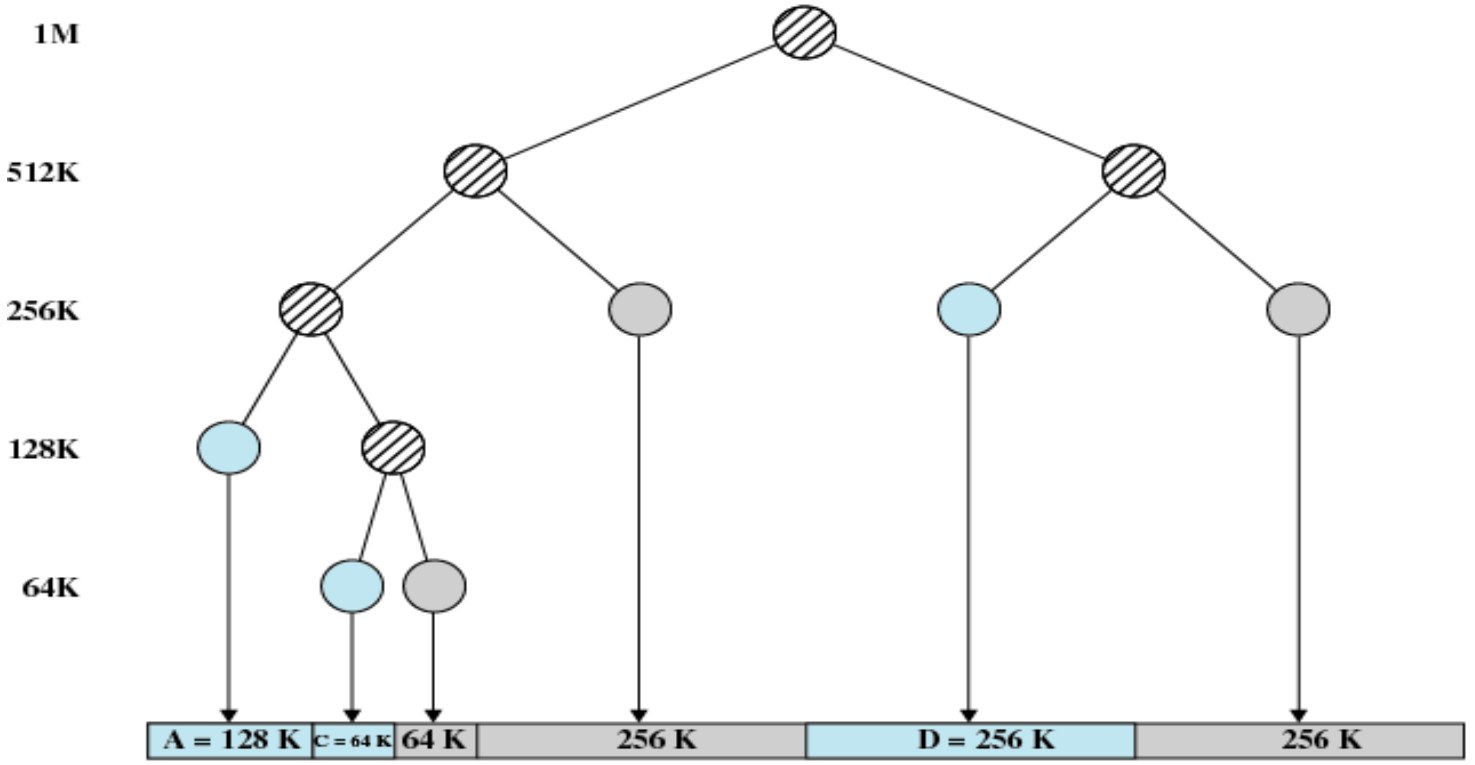
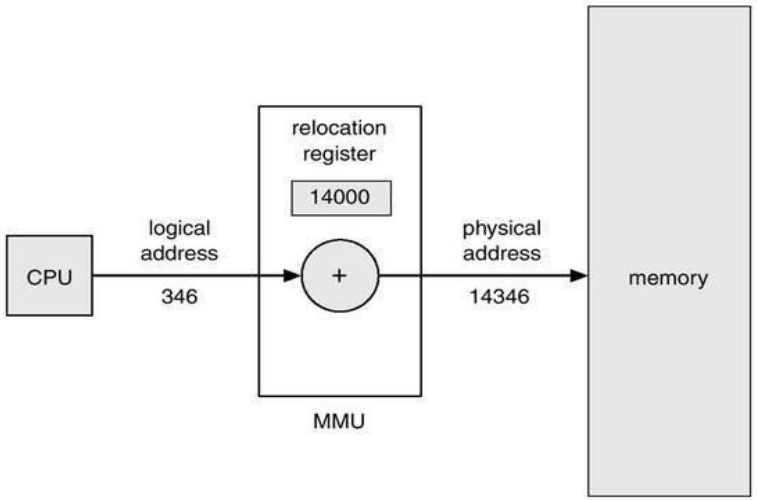


Figure 7.7 Tree Representation of Buddy System

Memory Partitioning

- Relocation(재배치)
  - ✓ Several type of Addresses
    - Logical address: 명령어와 데이터 위치는 스왑인되거나 이동될 때마다 변경될 수 있으므로, 현재 데이터가 로드된 메모리와는 독립적인 메모리 위치에 대한 참조로서, 실제 메모리 접근을 하려면 물리주소로 변환되어야 함
      - Relative address(상대주소)
        - 어떤 알려진 지점, 주로 CPU의 한 레지스터 값으로부터 상대적인 위치를 나타내는 주소
        - 동적수행시간 적재기법 사용
        - (particular example of logical address)
    - Physical address 또는 absolute address : 주기억장치(main memory) 내에서의 실제 위치



Memory Partitioning

- 상대주소는 CPU에 의해 2가지 처리과정
  - ✓ 베이스 레지스터 값과 이 상대주소 값을 더해 절대주소로 변환
  - ✓ 이 절대주소와 경계(bound) 레지스터 값을 비교하여
    - 현재 주소가 경계내에 있으면 명령이 실행되고, 그렇지 않으면 OS로 인터럽트 발생되어 에러 처리!

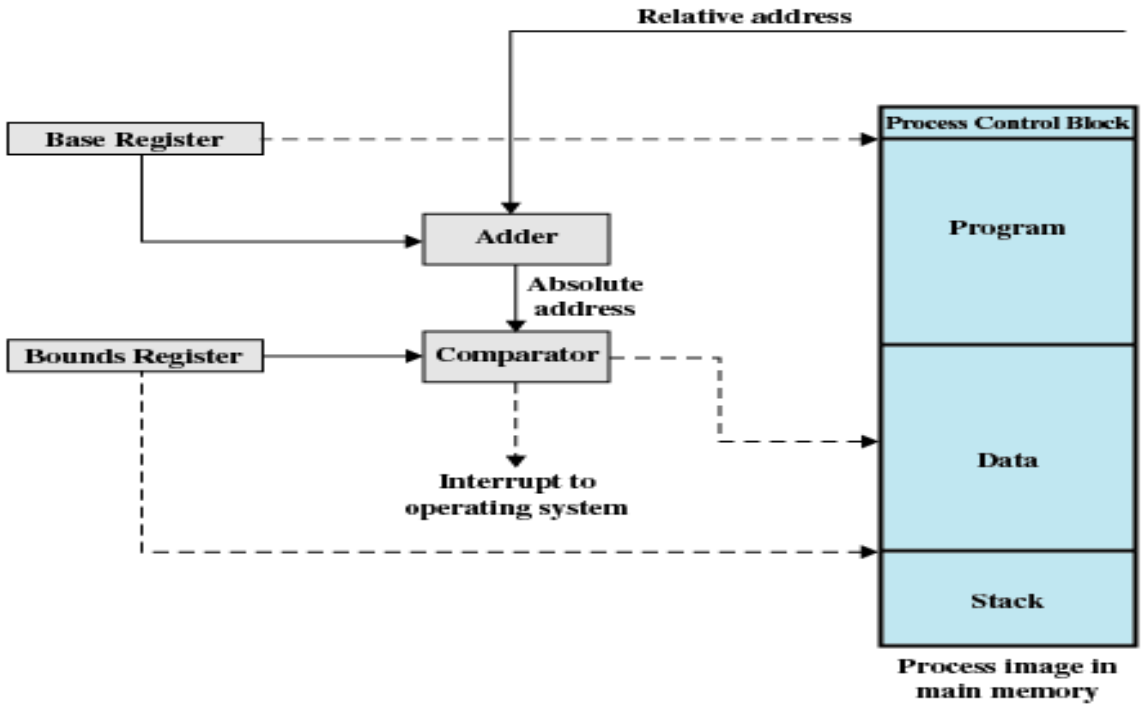


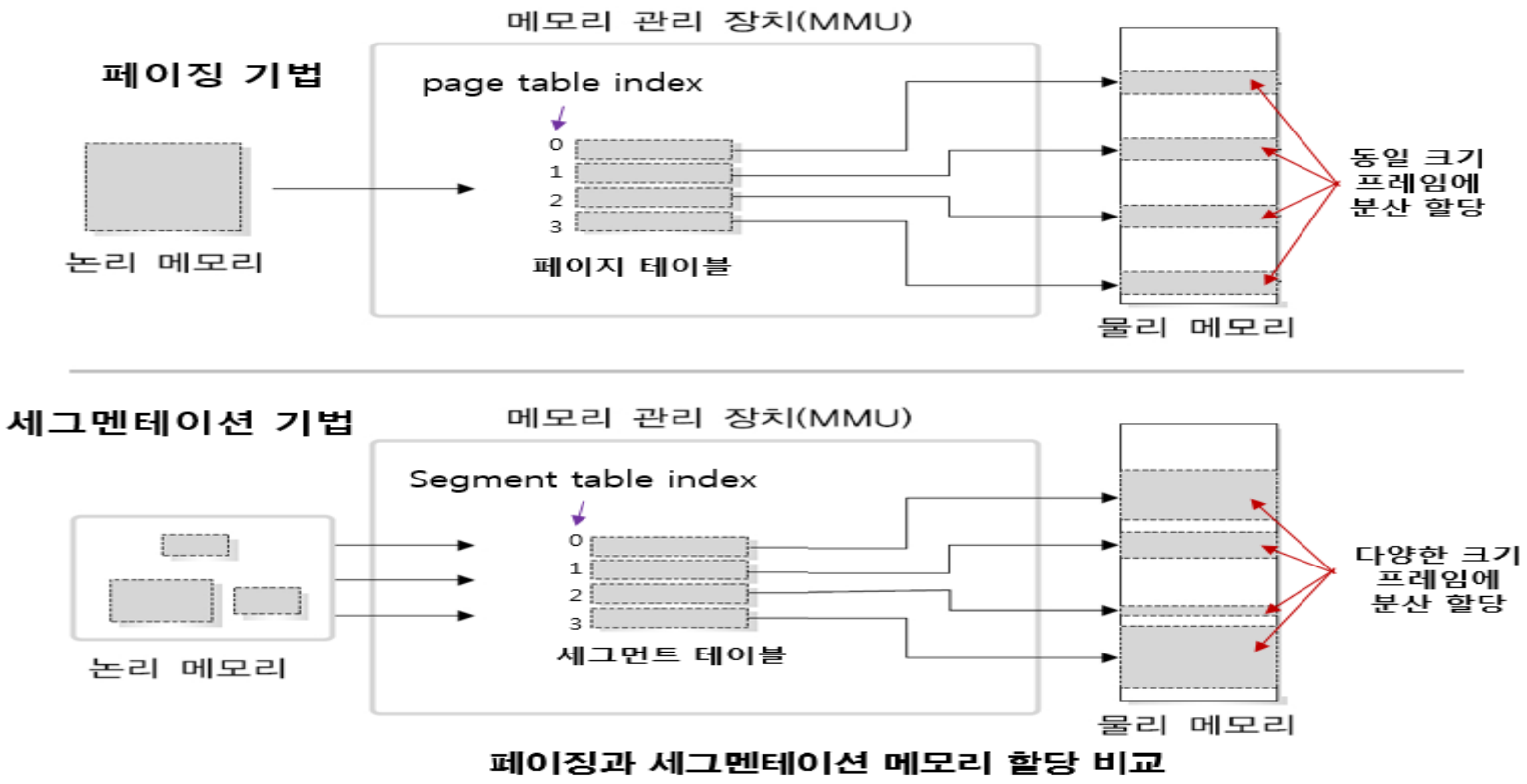
Figure 7.8 Hardware Support for Relocation

7.3 페이징 (paging)

- 페이징
  - ✓ 페이지(page) : 작은 고정 사이즈의 프로세스 조각
  - ✓ 프레임(frame) : 페이지와 크기가 같은 주 기억장치 메모리 조각
  - ✓ 페이지 테이블 : 프로세스의 각 페이지에 해당하는 프레임 위치 관리
  - ✓ 외부 단편화 발생 안함
  - ✓ 내부 단편화 : 각 프로세스의 마지막 페이지에서만 발생
  - ✓ 단순 페이징 : 고정 분할 방법과 유사
- OS는 각 프로세스마다 하나의 page table 보유
  - ✓ 페이지테이블은 프로세스의 각 페이지들에 해당하는 frame 위치를 관리
  - ✓ 프로그램내에서 각 논리주소는 page #와 페이지내의 offset 으로 구성

7.3 페이징 (paging)

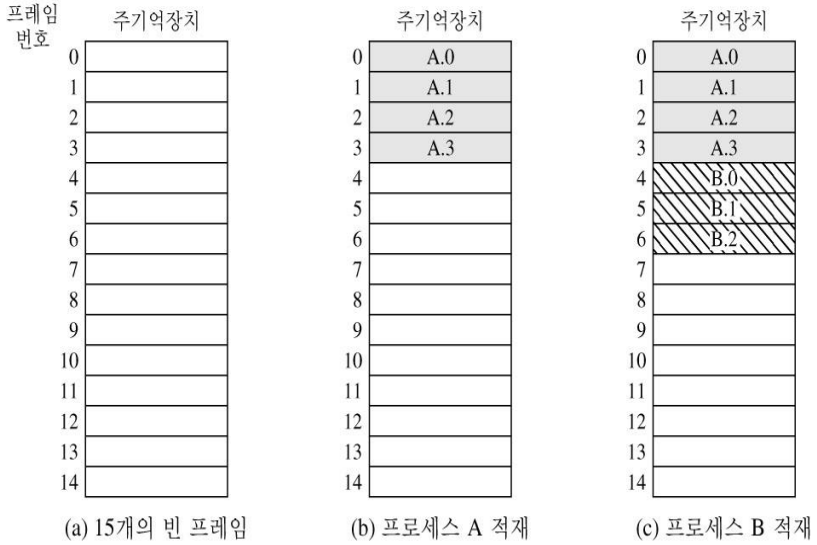
- 페이징과 세그멘테이션



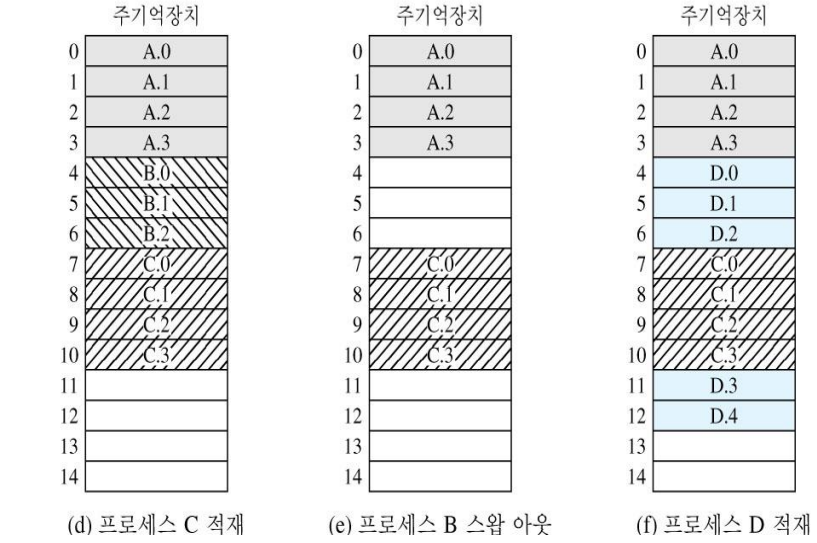
페이징과 세그멘테이션 메모리 할당 비교

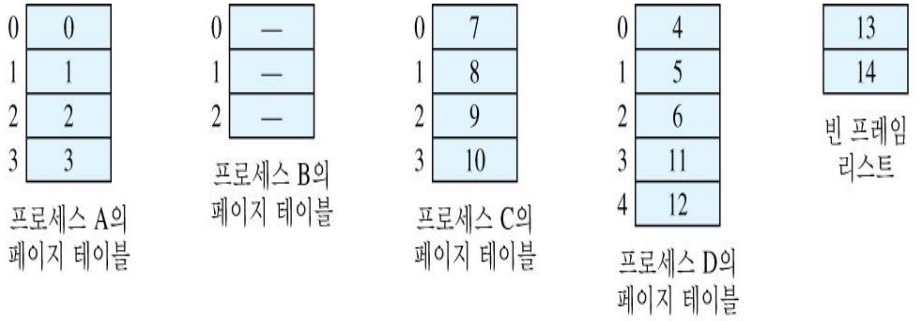
페이징 (paging)

- 페이징 기법의 예



<그림 7.9> 사용 가능한 빈 프레임에 프로세스 페이지 적재

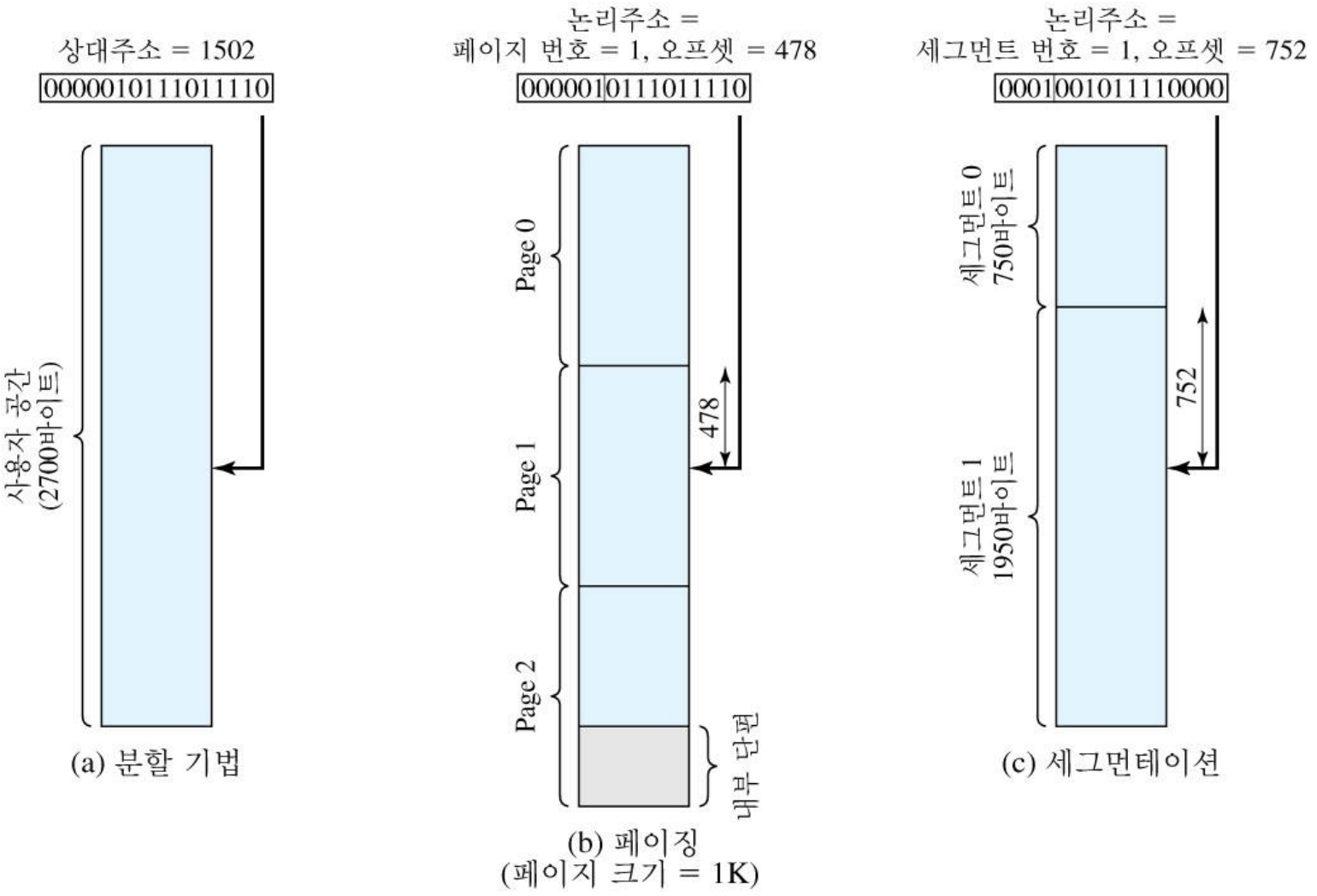




<그림 7.10> 그림 7.9의 예에서 (f) 시점의 자료 구조

페이징 (paging)

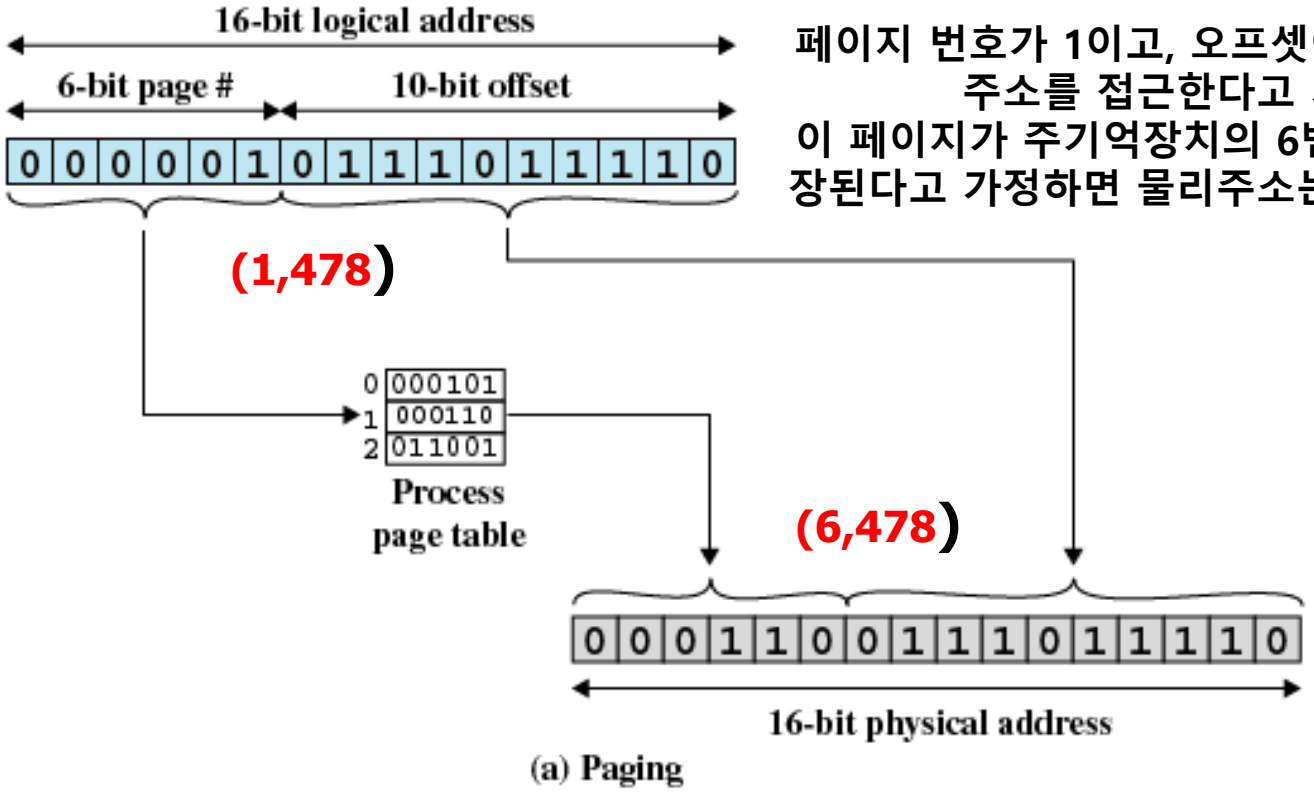
- 페이징 기법의 주소 관리=> (a),(b),(c) 상호변환가능



<그림 7.11> 논리 주소

Paging

- Translation
  - ✓ logical to physical address translation



(a) Paging

페이지 번호가 1이고, 오프셋이 478인 논리 주소를 접근한다고 가정.
이 페이지가 주기억장치의 6번 프레임에 저장된다고 가정하면 물리주소는 아래와 같음.

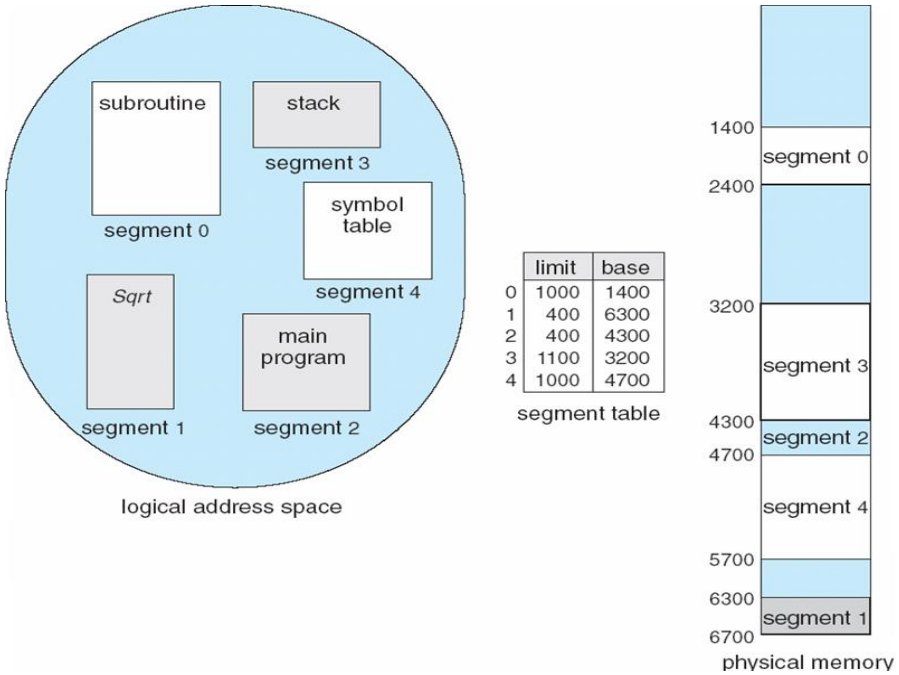
7.4 Segmentation

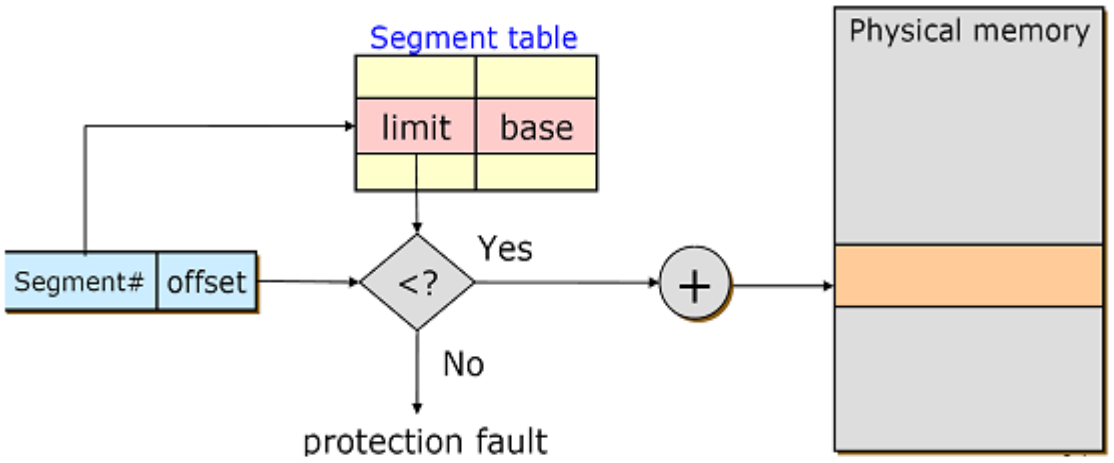
- Motivation
  - ✓ program is divided into a number of segments
  - ✓ All segments do not have to be of the same length
  - ✓ There is a maximum segment length

- Translation
  - ✓ segment table
  - ✓ each entry: base and length
  - ✓ logical address consists of two parts - a segment number and an offset

☞ Whereas paging is invisible to the programmer, segmentation is usually visible and is provided as a convenience for organizing program.

세그먼테이션 (segmentation)





Segmentation

- Translation Example
  - ✓ In this example, the maximum segment size is 4096 (12 bit)
  - ✓ In IA, segment and offset are managed in separated registers

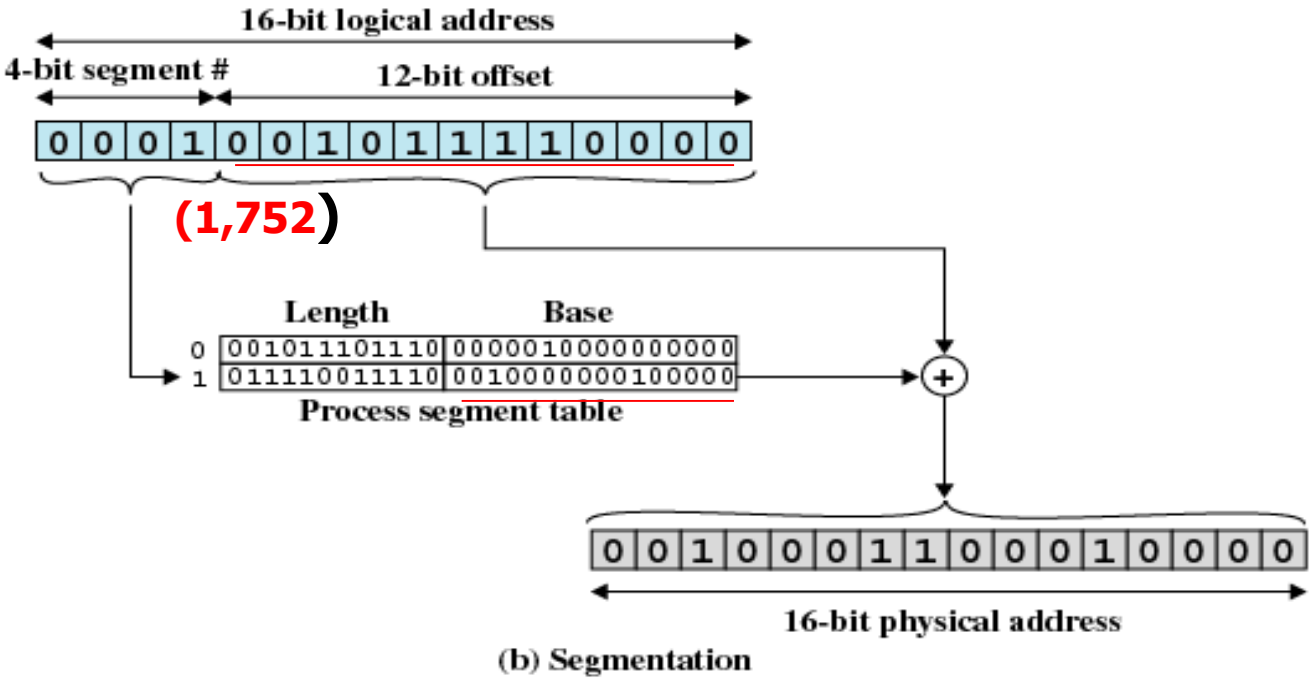


Figure 7.12 Examples of Logical-to-Physical Address Translation

프로세스가 메모리로 적재될 경우, 프로세스의 모든 세그먼트가 사용가능한 메모리 영역에 할당되어지고 그에 따라 세그먼트 테이블의 값이 설정

Summary

- Memory Management Requirements
- Memory Partitioning
  - ✓ fixed partitioning
  - ✓ dynamic partitioning
  - ✓ buddy
- Relocation
- Paging
  - ✓ page number, offset
  - ✓ page table: page number ➔ page frame
  - ✓ page fault
- Segmentation
  - ✓ segment number, offset
  - ✓ segment table: segment number ➔ base address, limit
  - ✓ segmentation fault (violation)

Appendix 7A Loading and Linking

- Linker and Loader: big picture

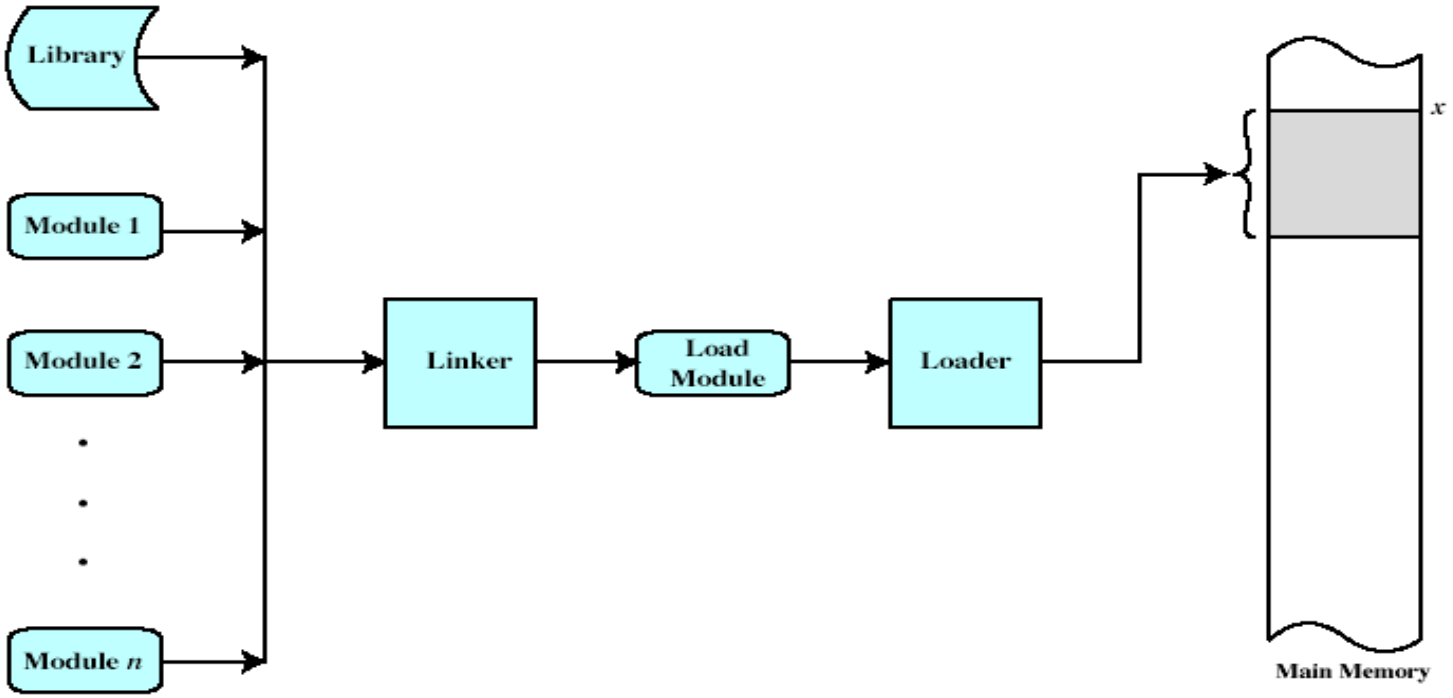


Figure 7.14 A Loading Scenario

☞ symbol resolving and address binding

Appendix 7A Loading and Linking

- Loading: new process image

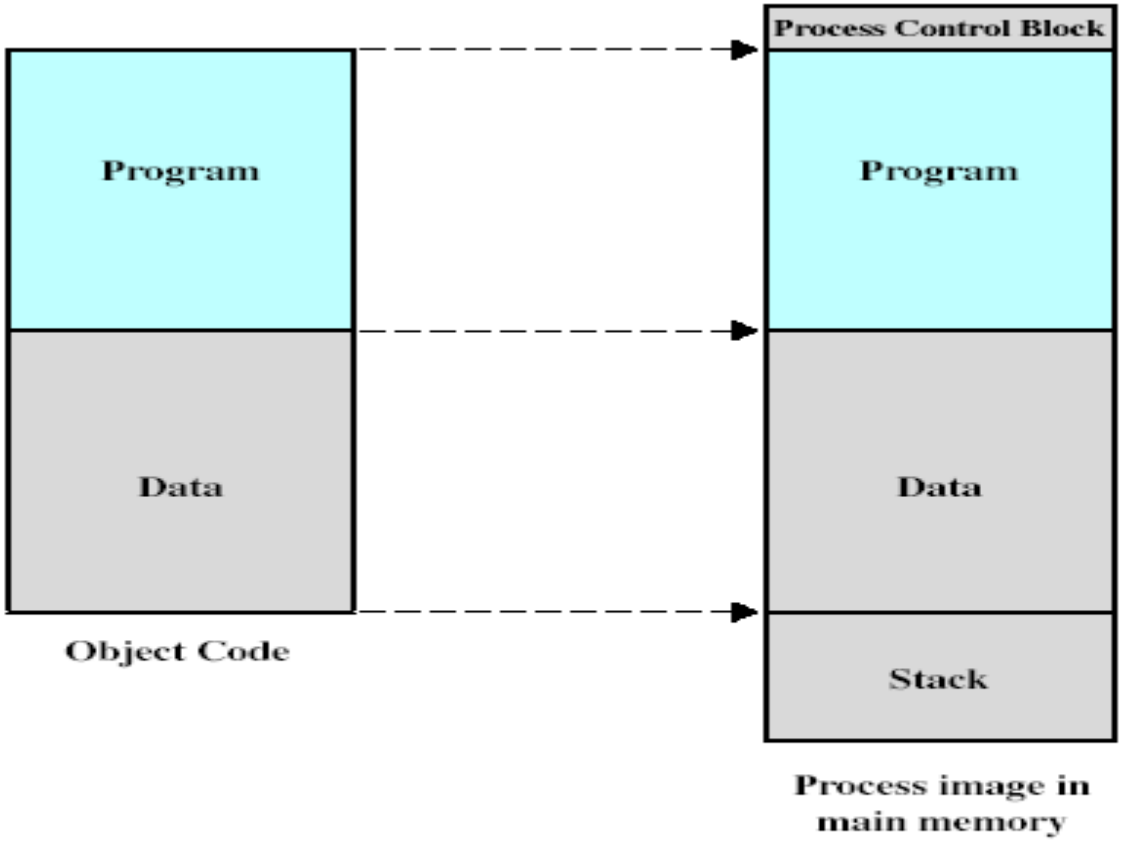


Figure 7.13 The Loading Function

Appendix 7A Loading and Linking

- Loading
  - ✓ absolute loading
  - ✓ relocatable loading(재배치가능 로딩)
  - ✓ dynamic run-time loading

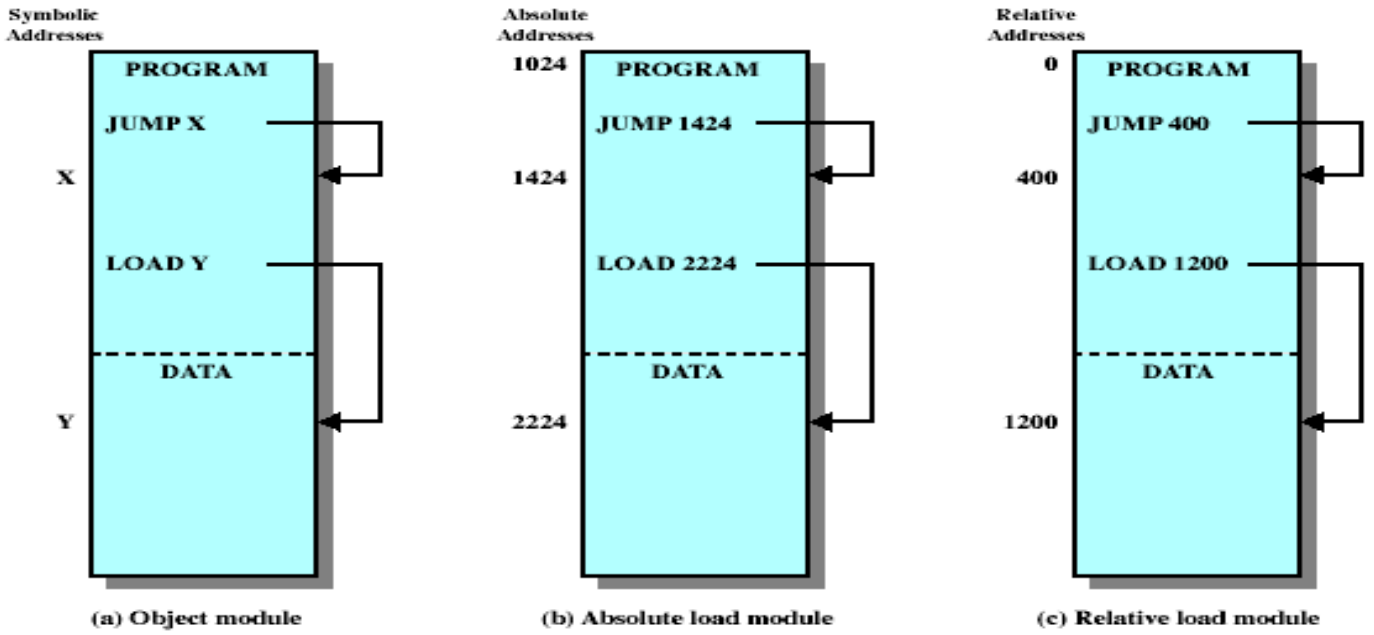


Figure 7.15 Absolute and Relocatable Load Modules

Appendix 7A Loading and Linking

- Loading
  - ✓ absolute loading
    - a module always is loaded into the same location in main memory
    - binding: programming time, compile or assembly time
    - any modification of program causes all of the address to be altered
  - ✓ relocatable loading
    - a module that can be located anywhere in main memory
    - binding: loading time
  - ✓ dynamic run-time loading
    - different memory location during execution (swap, compaction)
    - binding: run time
    - H/W support

Binding Time	Function
Programming time	All actual physical addresses are directly specified by the programmer in the program itself.
Compile or assembly time	The program contains symbolic address references, and these are converted to actual physical addresses by the compiler or assembler.
Load time	The compiler or assembler produces relative addresses. The loader translates these to absolute addresses at the time of program loading.
Run time	The loaded program retains relative addresses. These are converted dynamically to absolute addresses by processor hardware.

Appendix 7A Loading and Linking

- Linking
  - ✓ 링커의 기능은 여러 목적 모듈을 입력으로 하여 로더에게 넘겨주기 위한 적재모듈 생성.이 적재모듈은 여러 프로그램과 데이터가 통합된 형태
  - ✓ resolve external reference

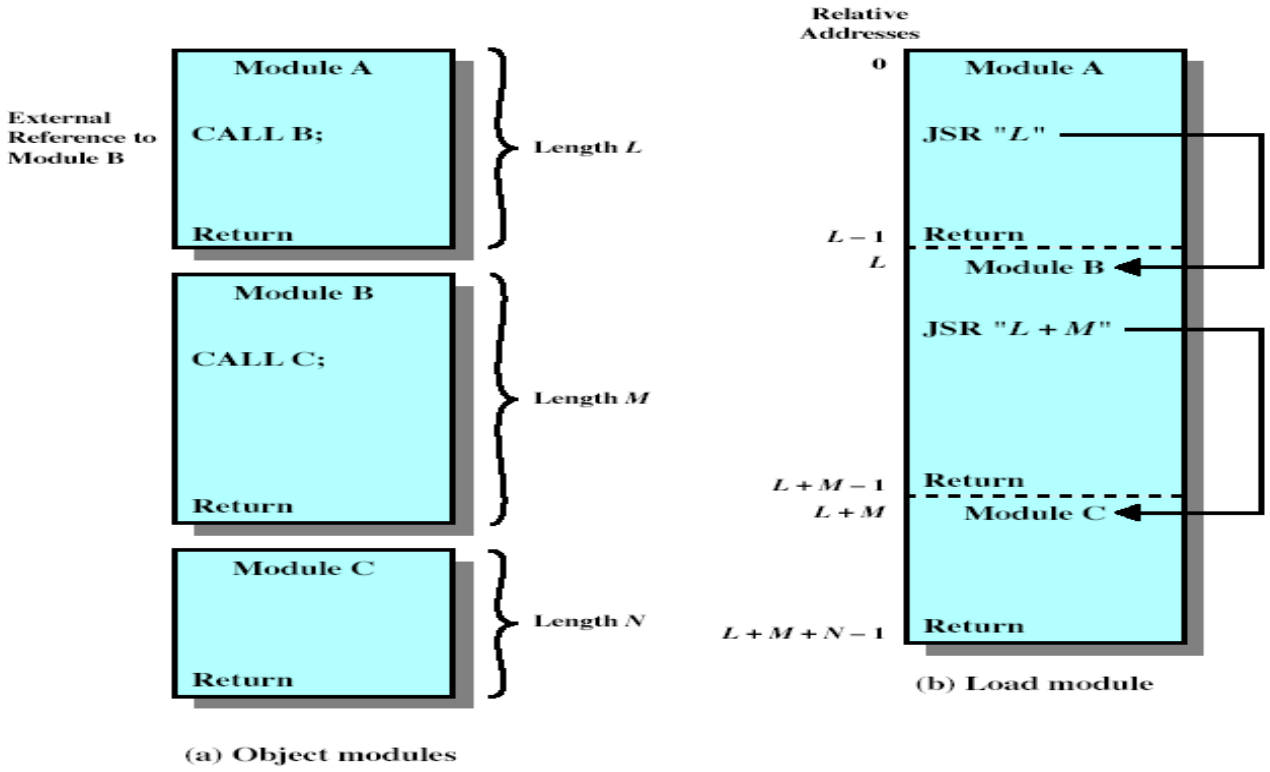


Figure 7.16 The Linking Function

Appendix 7A Loading and Linking

- Dynamic Linker
  - ✓ 외부 모듈과의 링킹 작업을 적재모듈이 생성된 이후로 연기. 그러므로 적재모듈에는 다른 프로그램을 참조하는 부분이 해결되지 않은 채 존재할 수 있으며, 이러한 참조는 적재시점 또는 실행시점에 해결될 수 있음. (static link vs dynamic link)
  - ✓ load-time dynamic linking
    - During loading, any reference to an unresolved external module (target module) causes the loader to find the target module, load it, and alter the reference to the address
    - advantage: upgrading, sharing, extensibility
  - ✓ run-time dynamic linking
    - When a call is made to an absent external module (target module), the operating system locates the module, load it, and links it to the calling module
    - advantage: memory efficiency, content dependent invocation, flexibility, interoperability

애니메이션

- http://williamstallings.com/OS/Animations.html

  - ✓ Overlays for primitive memory management
  - ✓ Dynamic relocation using a relocation register
  - ✓ Multiple-partition contiguous memory allocation
    Compaction
  - ✓ Paging hardware
  - ✓ Paging model of logical and physical memory
  - ✓ Paging Example for a 32-byte memory with 4-byte pages
  - ✓ Segmentation hardware
  - ✓ Paged segmentation