운 영 체 제

3장 기억장치 관리(1)

3장 기억장치 관리

3.1 기억장치 관리의 개요

- 기억장치 관리의 발전
- 기억장치 관리의 주소 바인딩

3.2 기억장치의 계층 구조 및 기억장치의 관리 기법

- 인출(Fetch) 기법
- 배치(Placement) 기법
- 교체(Replacement) 기법

3.3 주기억장치 할당 기법(단일 사용자 연속 기억장치 할당)

3.4 주기억장치 할당 기법(고정 분할 기억장치 할당)

3.5 주기억장치 할당 기법(가변 분할 기억장치 할당)

3.6 기억장치 교체(swapping)

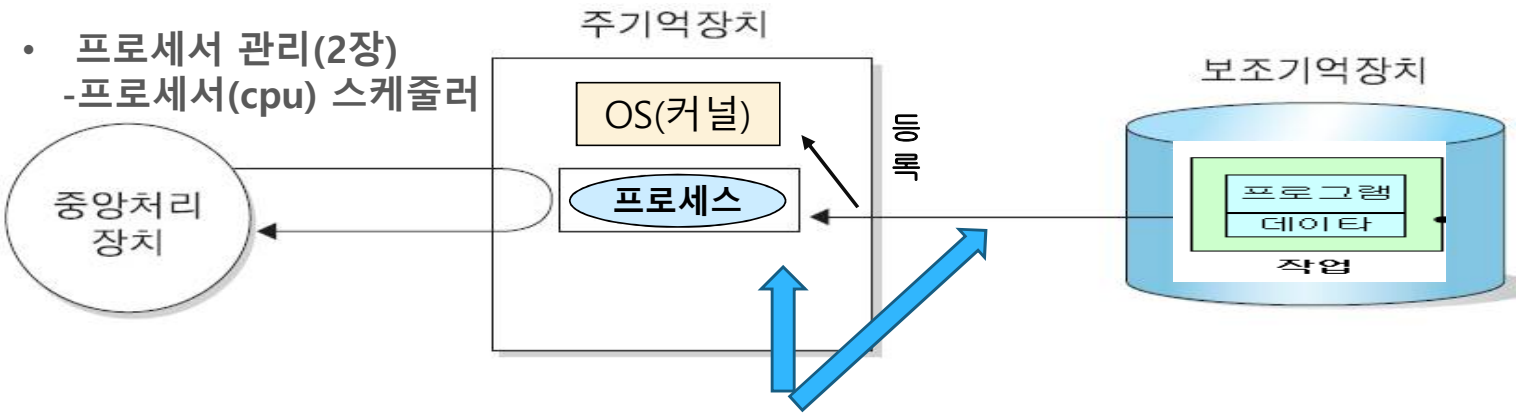
주기억장치 관리 기법의 문제점과 해결 방법(단편화=>통합,압축)

학습 내용

3.1 기억장치 관리의 개요

- 기억장치의 계층 구조
- 기억장치 관리의 발전
- 기억장치 관리의 주소 바인딩
- 기억장치 구성 정책
- 기억장치의 관리 기법
  - 인출(Fetch) 기법
  - 배치(Placement) 기법
  - 교체(Replacement) 기법
  - 할당(allocation) 기법

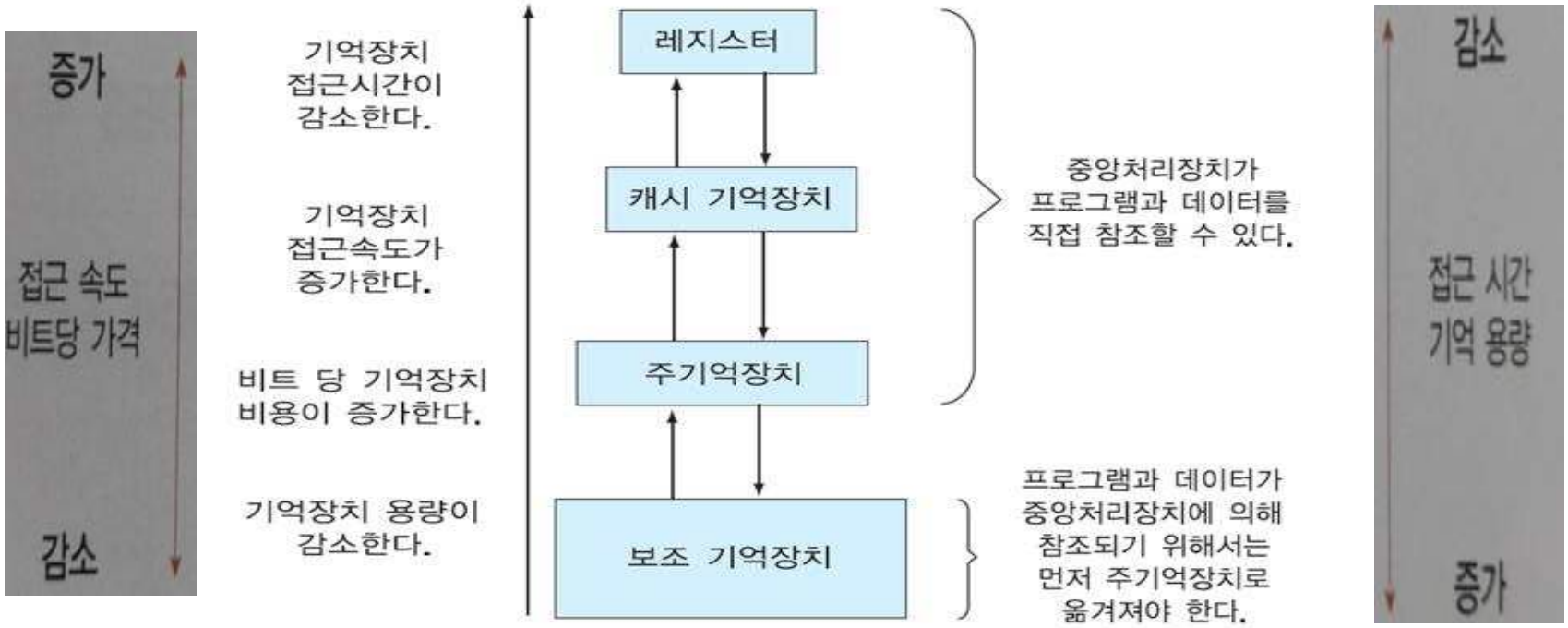
3장 기억장치 관리의 개요



- 프로세서 관리(2장)
  -프로세서(cpu) 스케줄러
- **주기억장치의 관리(3장)**
  -Job 스케줄링
  -주소 바인딩
  -**주기억장치 관리 기법**
    -인출 기법(요구 인출 기법,예상인출 기법)
    -배치 기법(최초 적합(first-fit), 최적 적합(best-fit) 및 최악 적합(worst-fit))
    -교체 기법
  -**주기억장치 할당 기법**
    -**연속 할당 기법(단일사용자 연속, 고정 분할, 가변 분할)**
- 가상 메모리 관리(4장)
  -분산 할당 기법(페이징 기법, 세그먼테이션 기법)

기억장치의 계층 구조

- 프로그램과 데이터가 실행되거나 참조되기 위해서는 주기억장치에 있어야 한다.



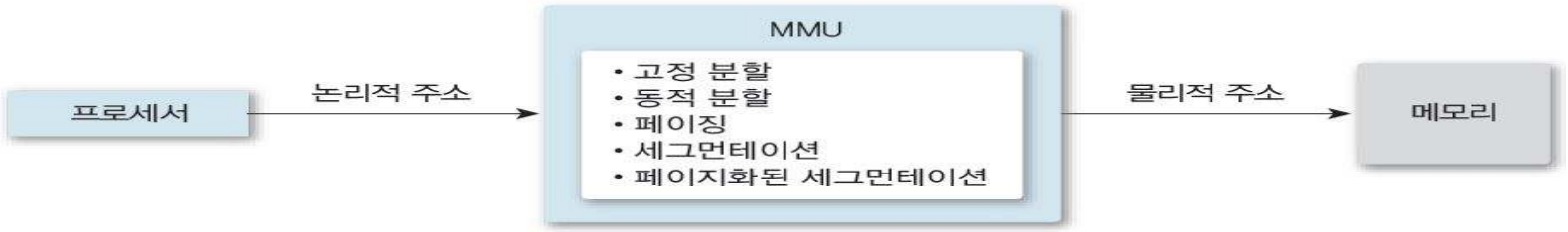
기억장치의 계층적 구조

기억장치 관리의 개요

- 프로그램과 데이터는 직접 실행되거나 참조되기 위해서는 주기억장치 내에 있어야 함
- 기억 장치는 프로그램을 실행하는 중요한 작업 공간
- 주기억장치는 용량이 제한되어 있고 값이 비싸다는 단점
- **기억 장치 관리의 개념**
  - 기억 장치 관리는 프로세스들을 위해 메모리 할당, 제거, 보호하는 활동
  - 프로세스의 요청에 따라 기억 장치의 일부를 할당하고 필요 없으면 자유로이 재사용 할 수 있도록 하는 것
  - 디스크에 있는 프로그램을 실행하려면 먼저 기억 장치에 적재 후 기억장치 관리자가 예약된 메모리 할당해 주는 것
  - 다중 프로그래밍 시스템에서 여러 프로세스가 기억 장치에 상주할 수 있도록 운영체제가 동적으로 메모리 세분화
  - 기억 장치 관리자는 기억 장치와 관련된 여러 정책을 수립하고 정책에 따라 기억 장치를 관리
- **기억 장치 관리 정책**
  - 적재 정책, 배치 정책, 대치 정책 등

기억장치 관리의 개요

- **기억 장치 관리하는 방법을 알려면 기억 장치의 구조 이해**
  - 기억 장치는 주소들의 연속이고 주소는 프로그래머 관점의 논리적 주소와 저장 공간 관점의 물리적 주소로 볼 수 있다
- **기억 장치의 주소 변환**
  - **논리적 주소와 물리적 주소의 변환은 기억 장치 관리자(MMU:메모리관리장치)가 처리**
  - 주소 변환 방법은 고정 분할, 동적 분할, 페이징, 세그먼테이션 등 변환 방법 사용
- **기억 장치의 주소 바인딩**
  - 프로세서는 논리적 주소에 대응하는 물리적 주소를 알아야 프로세스를 실행하므로 두 주소를 연결해 주는 사상(mapping) 작업 즉 **주소 바인딩(binding)**이 필요



메모리관리장치의 주소 변환(논리적 주소 → 물리적 주소)

기억장치 관리의 발전

기억장치 관리의 발전

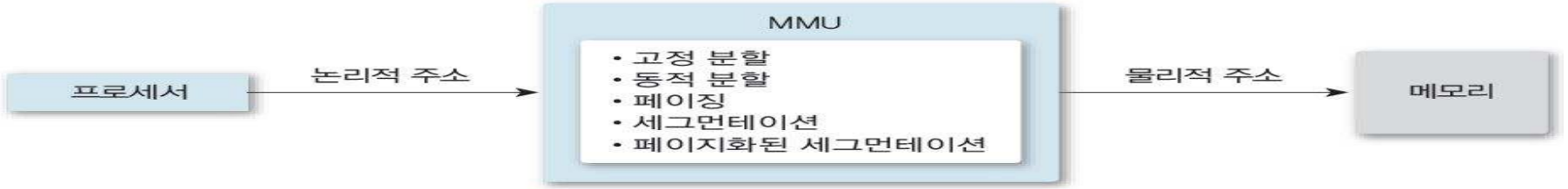
<table>
<tr><td colspan="2">연속 적재 기법
(Contiguous Loading)</td><td colspan="7">분산 적재 기법
(Scatter Loading)</td></tr>
<tr><td colspan="6">실기억 공간
(Real Memory)</td><td colspan="3">가상 기억 공간
(Virtual Memory)</td></tr>
<tr><td colspan="2">단일 사용자
(Single User)</td><td colspan="7">다중 프로그래밍
(Multi-Programming)</td></tr>
<tr><td>오버레이
Overlay</td><td>교체
Swapping</td><td>고정 분할
Fixed Partition</td><td>동적 분할
Dynamic Partition</td><td>페이징
Paging</td><td>세그먼테이션
Segmentation</td><td>페이지/세그먼트의 혼합 형태</td><td>요구 페이징</td><td>요구 세그먼테이션</td></tr>
</table>

기억장치 관리의 주소 바인딩

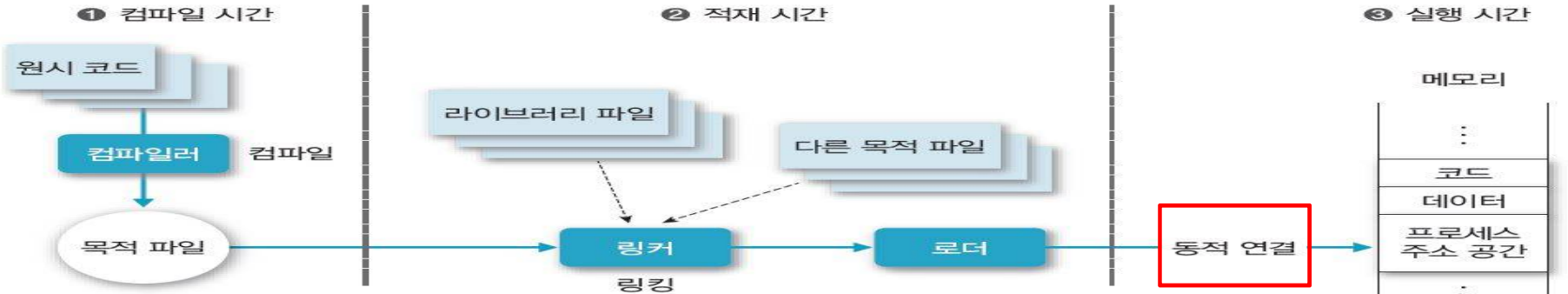
- **주소 바인딩(address binding)**
  - 정의:프로세스를 실행하기 위해 논리적 주소(logical address)를 물리적 주소로 사상(mapping)
- **주소 바인딩 시점에 따라 (2장 프로세스 관리 링커와 로더 참고)**
  - 컴파일 시간(compile time) 바인딩
    - 프로세스가 메모리에 적재될 위치를 컴파일 과정에서 알 수 있다면 컴파일러는 물리적 주 소 생성 가능 => 프로그래머에 의해 연결 (예: 로더의 종류 : 컴파일 즉시 로더)
  - 적재 시간(load time) 바인딩
    - 적재될 위치를 컴파일러가 알지 못하면 컴파일러는 상대 주소 생성, 시작 주소가 변하면 단지 변화된 값을 반영하려고 사용자 코드를 재배치
    - 주소 바인딩은 프로그램이 기억장치에 적재되는 시간에 적재기(loader)에 의해 이루어짐 (예: 로더의 종류 : 정적 재배치 로더)
  - 실행 시간(execution time) 바인딩
    - 실행되는 동안에 기억장치의 한 세그먼트에서 다른 세그먼트로 옮겨질 경우 주소 바인딩은 실행시간에 이루어 짐 (현재 범용 운영체제 대부분 실행 시간에 바인딩 방법 사용)

기억장치 관리의 주소 바인딩

- 주소 바인딩 **(2장 프로세스 관리 : 시스템소프트웨어 종류 링커와 로더 참고)**
  - 논리적 주소와 물리적 주소의 연결, 매핑 시켜 주는 작업(주소 바인딩 (address binding))



메모리관리장치의 주소 변환(논리적 주소 → 물리적 주소)



❶ 컴파일 시간 바인딩(컴파일 즉시 로더)
프로세스가 메모리에 적재될 위치를 컴파일 과정에서 알 수 있다면 컴파일러는 물리적 주소 생성 가능

❷ 적재 시간 바인딩(정적 재배치 로더)
프로세스를 메모리의 어디에 적재해야 할지 컴파일 과정에 알려 주지 않으면 컴파일러는 대체 가능한 상대 주소 생성. 시작 주소가 변하면 단지 변화된 값을 반영하려고 사용자 코드를 재적재(정적 대치)

❸ 실행 시간 바인딩(동적 로더)
프로세스 실행 도중 메모리의 한 세그먼트에서 다른 세그먼트로 이동 한다면 바인딩은 수행 시간까지 연기(지연). 이런 주소 체계는 기본 및 경계(한계) 레지스터 등 특수 하드웨어의 지원 필요. 현재 범용 운영체제 대부분 실행 시간에 바인딩 방법 사용 .

기억장치 구성 정책

- 조건
  - 프로세스의 실행에 필요한 데이터나 스택 등을 포함한 사용자 프로그램 내용 전체가 주기억장치에 연속으로 적재되어 실행되는 환경 일 때
- 주기억장치 구성 정책
  - 주기억장치를 동시에 할당 받을 수 있는 프로세스의 수
    - 동시에 사용자 프로그램이 주기억 장치에 적재 가능한 수
    - 단일 프로그래밍, 다중 프로그래밍
  - 각 프로세스에게 할당되는 주기억장치의 양
    - 다중프로그래밍 정도가 2 이상인 경우 동일한 양으로 또는 다른 양으로 할당할 지의 여부
      - 분할영역 (partition)의 크기를 같게/다르게 설정
  - 주기억장치 분할 방법
    - 다중프로그래밍 정도가 2 이상인 경우 분할 형태를 이후 변형여부 설정
      - 고정 분할(정적 분할), 가변 분할(동적 변할)
  - 각 프로세스에게 할당된 분할영역의 교체 가능성
    - 실행중인 프로세스가 할당된 영역만 사용할 것인지 아니면 할당 되지 않은 영역도 사용할 수 있도록 하는 정책
  - 프로세스에게 할당되는 주기억장치 영역의 연속성
    - 연속 할당 정책/비연속 할당 정책

기억장치의 관리 기법

- **주기억장치 관리 기법**
- **값비싼 주기억장치 자원을 가장 효율적으로 이용하기 위한 방안으로 여러 기법이 있다.**
- **기억장치의 관리전략은 보조기억장치의 프로그램이나 데이터를 주기억장치에 적재시키는 시기, 적재위치 등을 지정하여 한정된 주기억장치 공간을 효율적으로 사용하기위한 것으로 인출전략, 배치전략, 교체전략이 있다.**
  - 인출 기법 (fetch strategy) (적재할 시기 결정, **언제 할당**)
    - 새로 생성된 프로세스에 대한 주기억장치 할당 시기 결정, 호출=적재=인출
    - 주기억장치에 적재할 다음 프로그램이나 데이터를 언제 가져올 것인가를 결정
    - 요구 인출(demand fetch) 기법, 예상 인출(anticipatory fetch) 기법
  - 배치 기법 (placement strategy) (공간의 위치 결정, 어떤 메모리 블록을 할당)
    - 주기억장치를 할당 받고자 하는 프로세스에게 할당 공간(메모리 블록)의 **위치 결정**
    - 새로 인출된 데이터나 프로그램을 주기억장치의 어디에 위치시킬 것인가를 결정하는 기법
    - 최초 적합(first-fit), 최적 적합(best-fit) 및 최악 적합(worst-fit)
    - **단편화**
  - 교체 기법(replacement strategy) (교체 대상 프로세스 결정, **어떤 프로세스와 교체**)
    - 주기억장치 공간 부족 시 어느 프로세스의 공간을 선점할지 결정, 교체=대치=재배치
    - 새로 들어온 프로그램이 들어갈 장소를 마련하기 위해서 어떤 프로그램 및 어떤 데이터를 제거할 것인가를 결정 함
  - **할당 기법 (allocation strategy) =>** 주기억장치 할당량 결정
    - 새로 생성되는 프로세스에 대한 공간 할당량 결정

기억장치의 관리 기법

- **기억장치 관리 기법**
  - 인출(fetch) 기법 (적재할 시기 결정, **언제 할당**)
    - 주기억장치에 적재할 다음 프로그램이나 데이터를 언제 가져올 것인가를 결정
    - 보조기억장치에 보관중인 프로그램이나 데이터를 언제 주기억장치로 적재할 것인지를 결정하는 전략이다.
    - 요구 반입(Demand fetch) : 실행중인 프로그램이 특정프로그램이나 데이터 등의 참조를 요구할 때 적재하는 방법
    - 예상반입(Anticipatory fetch) : 실행중인 프로그램에 의해 참조될 프로그램이나 데이터를 미리 예상하여 적재하는 방법

기억장치의 관리 기법

- **기억장치 관리 기법**
  - 교체(replacement) 기법(교체 대상 프로세스 결정, 위치 결정)
    - 새로 들어온 프로그램이 들어갈 장소를 마련하기 위해서 어떤 프로그램 및 어떤 데이터를 제거할 것인가를 결정 함
    - 주기억 장치의 모든 영역이 이미 사용중인 상태에서 새로운 프로그램이나 데이터를 주기억 장치에 배치하려고 할 때, 이미 사용되고 있는 영역 중에서 어느 영역을 교체하여 사용할 것인지를 결정하는 전략이다.

    - 교체전략에는 FIFO, OPT, LRU, LFU, NUR, SCR 등이 있다.
    - => 가상 기억 장치 교체 기법과 동일(4장에서)

기억장치의 관리 기법

- **기억장치 관리 기법**
  - 배치(placement) 기법(공간의 위치 결정)
    - 새로 인출된 데이터나 프로그램을 주기억장치의 어디에 위치시킬 것인가를 결정하는 기법
    - **최초 적합(first-fit), 최적 적합(best-fit) 및 최악 적합(worst-fit)**

최초 적합(First Fit)	프로그램이나 데이터가 들어갈 수 있는 크기의 빈 영역 중에서 첫 번째 분할 영역에 배치시키는 방법
최적 적합(Best Fit)	프로그램이나 데이터가 들어갈 수 있는 크기의 빈 영역 중에서 단편화를 가장 적게 남기는 분할 영역에 배치시키는 방법
최악 적합(Worst Fit)	프로그램이나 데이터가 들어갈 수 있는 크기의 빈 영역 중에서 단편화를 가장 많이 남기는 분할 영역에 배치시키는 방법

    - **단편화** : 주기억장치의 분할된 영역에 프로그램이나 데이터를 할당할 경우 분할된 영역이 프로그램이나 데이터보다 작거나 커서 생기는 빈 기억공간을 의미

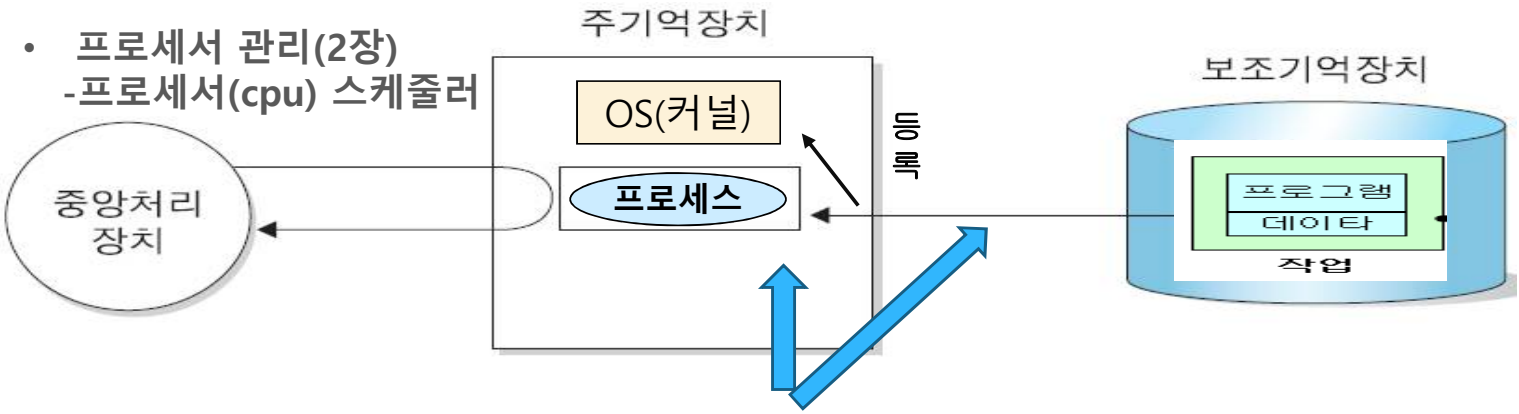
기억장치의 관리 기법

- **기억장치 관리 기법**
  - 배치(placement) 기법(공간의 위치 결정)
  - 예제) 주기억장치 상태가 다음 표와 같다. 기억장치 관리 전략으로 first fit, best fit, worst fit 방법을 사용하려 할때, 각 방법에 대하여 10K의 프로그램이 할당 받게 되는 영역의 번호는?

영역번호	영역크기	상태
1	5K	공백
2	14K	공백
3	10K	사용중
4	12K	공백
5	16K	공백

- First fit :
- Best fit :
- Worst Fit:

3장 기억장치 관리의 개요



- 프로세서 관리(2장)
  -프로세서(cpu) 스케줄러
- **주기억장치의 관리(3장)**
  - **-Job 스케줄링**
  - **-주소 바인딩**
  - **-주기억장치 관리 기법**
    - -인출 기법(요구 인출 기법,예상인출 기법)
    - -배치 기법(최초 적합(first-fit), 최적 적합(best-fit) 및 최악 적합(worst-fit))
    - -교체 기법
  - **-주기억장치 할당 기법**
    - **-연속 할당 기법(단일사용자 연속, 고정 분할, 가변 분할)**
- 가상 메모리 관리(4장)
  - -분산 할당 기법(페이징 기법, 세그먼테이션 기법)

기억장치 관리의 발전

기억장치 관리의 발전

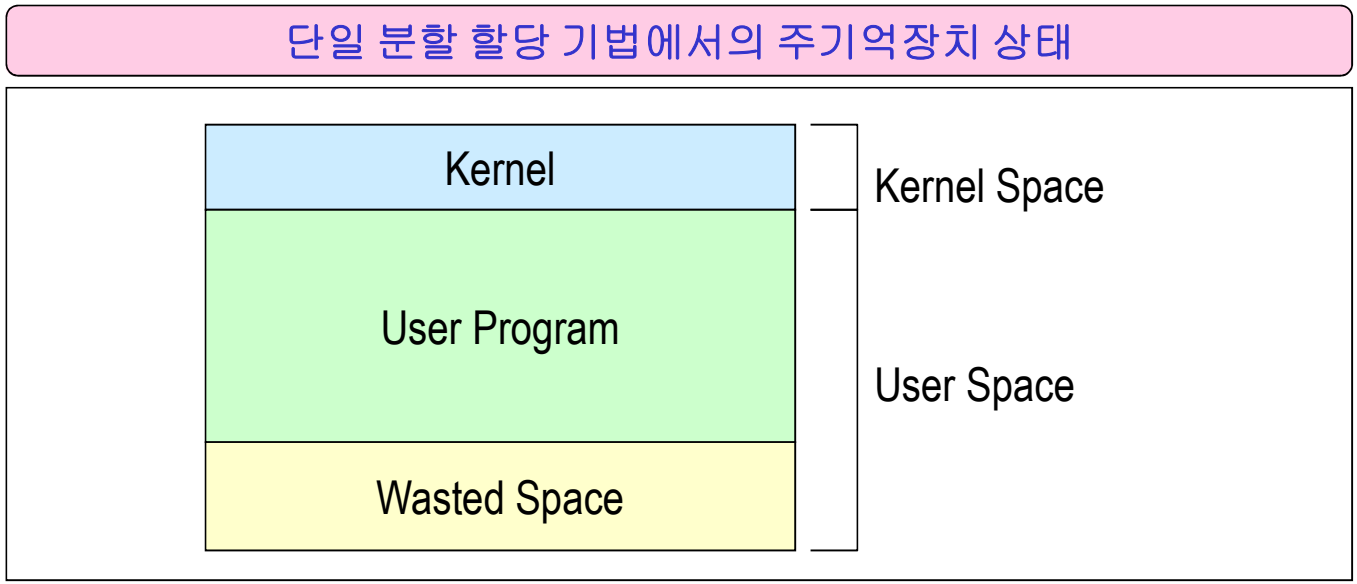
<table>
<tr><td colspan="2">연속 적재 기법
(Contiguous Loading)</td><td colspan="7">분산 적재 기법
(Scatter Loading)</td></tr>
<tr><td colspan="6">실기억 공간
(Real Memory)</td><td colspan="3">가상 기억 공간
(Virtual Memory)</td></tr>
<tr><td colspan="2">단일 사용자
(Single User)</td><td colspan="7">다중 프로그래밍
(Multi-Programming)</td></tr>
<tr><td>오버레이
Overlay</td><td>교체
Swapping</td><td>고정 분할
Fixed Partition</td><td>동적 분할
Dynamic Partition</td><td>페이징
Paging</td><td>세그먼테이션
Segmentation</td><td>페이지/세그먼트의 혼합 형태</td><td>요구 페이징</td><td>요구 세그먼테이션</td></tr>
</table>

기억장치 할당 기법

- 기억장치 할당 기법:
  - 주기억장치 할당 기법은 프로그램이나 데이터를 실행시키기 위해 주기억 장치에 어떻게 할당할 것인지에 대한 내용이며 , 연속할당기법과 분산할당기법으로 분류
  - 기억장치 할당량 결정(고정 크기 또는 필요한 만큼만 할당, 연속 또는 불연속)
  - 연속할당기법
    - 프로그램을 주기억장치에 연속으로 할당하는 기법
    - 단일분할 할당기법 : 오버레이 (overlay)기법, 스와핑(Swapping)기법
    - 다중분할 할당기법 : 고정(정적) 분할 할당 기법, 가변(동적) 분할 할당 기법
  - 분산할당기법
    - 프로그램을 특정단위의 조각으로 나누어 주기억장치 내에 분산하여 할당하는 기법
    - 페이징기법과 세그멘테이션 기법 **=> 4**장

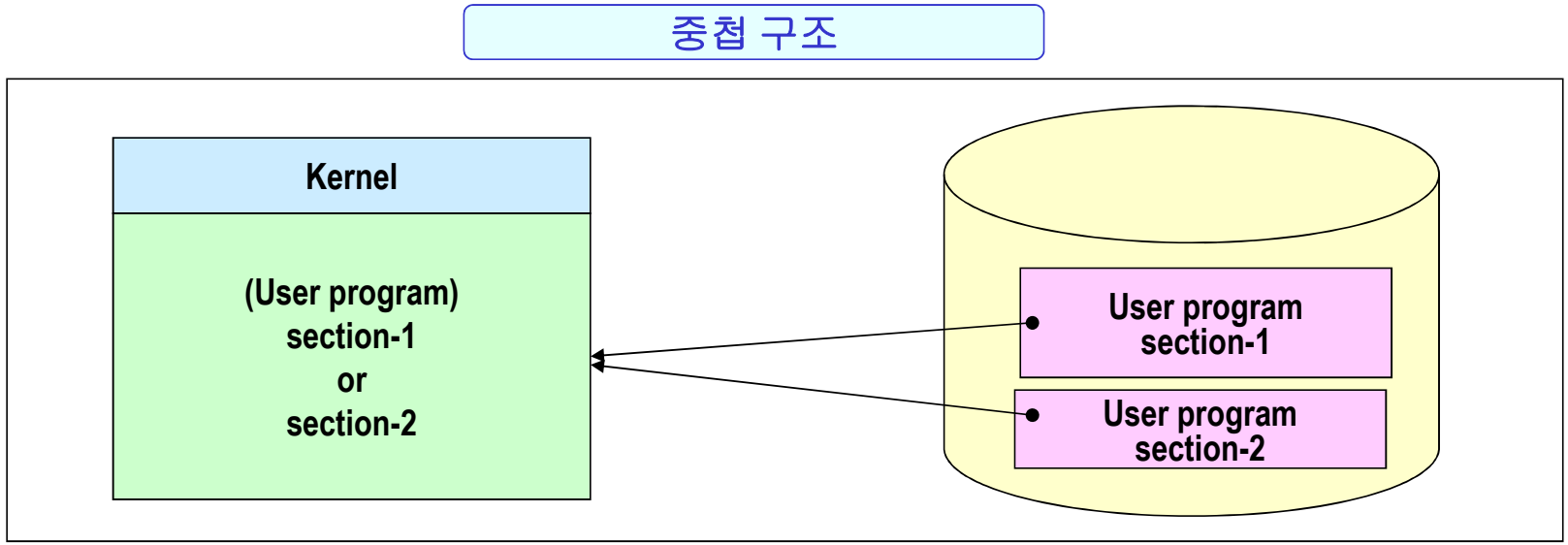
단일 분할 할당 기법(단일프로그래밍)

- 단일 분할 할당 기법 (단일프로그래밍:**uniprogramming** 시스템)
  - 항상 시스템 내에 하나의 프로세스만 존재
  - 단일 분할 할당 기법은 주기억장치를 운영체제 영역과 사용자 영역으로 나누어 한 순간에는 오직 한 명의 사용자만이 주기억장치의 사용자 영역을 사용하는 기법
  - 주기억장치 관리 기법이 매우 단순해 짐



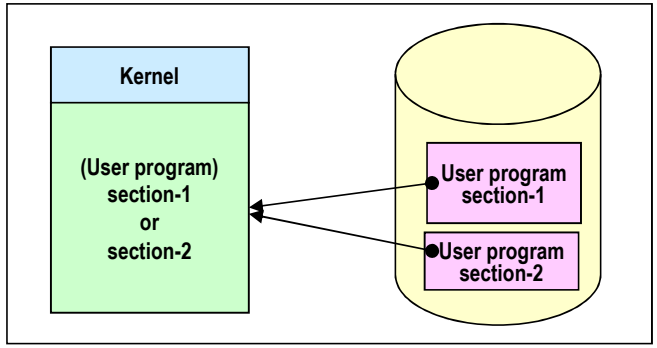
단일 분할 할당 기법(단일프로그래밍)

- 단일 분할 할당 기법 환경에서의 문제점(1)
  - 프로그램의 크기가 주기적장치의 가용 공간보다 클 경우
    - 해결
      - 중첩 구조 (overlay structure) 사용
      - 컴파일러 및 링커, 로더의 지원 필요



단일 분할 할당 기법(단일프로그래밍)

- 단일 분할 할당 기법 환경에서의 문제점(1)
  - 프로그램의 크기가 주기적장치의 가용 공간보다 클 경우
    - 해결
      - 중첩 구조 (overlay structure) 사용
      - 컴파일러 및 링커, 로더의 지원 필요

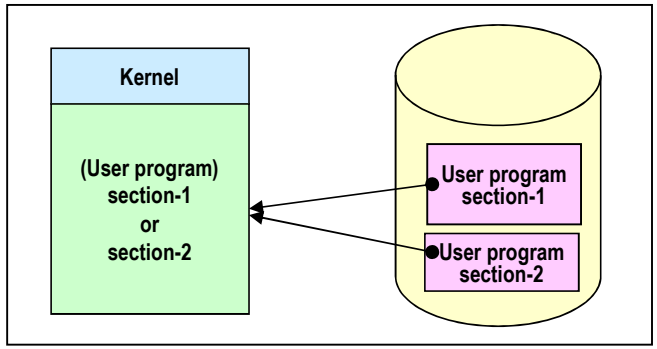


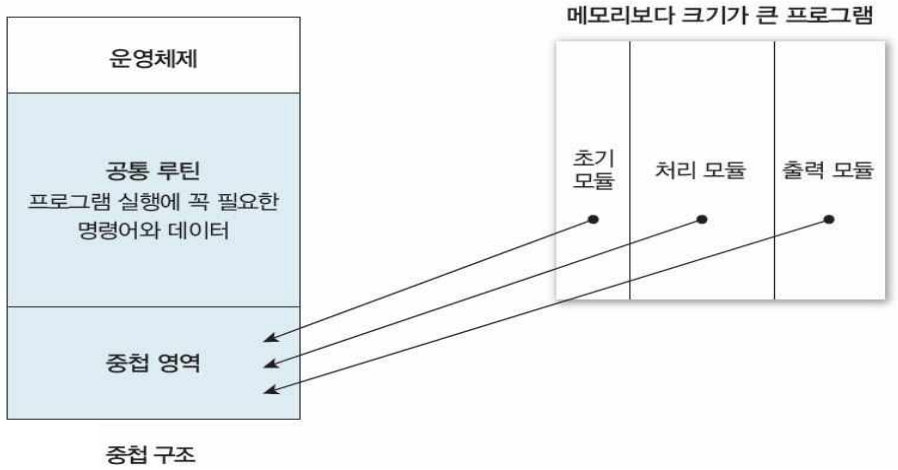
❖ **오버레이(Overlay)기법**

- 오버레이 기법은 주기억장치보다 큰 사용자 프로그램을 실행하기 위한 기법이다.
- 보조 기억장치에 저장된 하나의 프로그램을 여러 개의 조각으로 분할 한 후 필요한 조각을 차례로 주기억장치에 적재하여 프로그램을 실행한다.
- 프로그램이 실행되면서 주기억장치의 공간이 부족하면 주기억장치에 적재된 프로그램의 조각 중 불필요한 조각이 위치한 장소에 새로운 프로그램의 조각을 중첩(overlay)하여 적재한다.
- 프로그램을 여러 개의 조각으로 분할하는 작업은 프로그래머가 수행해야 하므로 프로그래머는 시스템 구조나 프로그램 구조를 알아야 한다.

단일 분할 할당 기법(단일프로그래밍)

- 단일 분할 할당 기법 환경에서의 문제점(1)
  - 프로그램의 크기가 주기적장치의 가용 공간보다 클 경우
    - 해결
      - 중첩 구조 (overlay structure) 사용
      - 당장 필요하지 않은 프로그램의 일부는 중첩으로 설정
  - 중첩 예

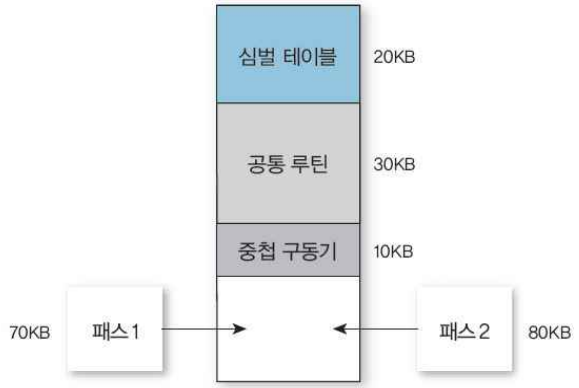




중첩 구조

- 패스 1 : 70KB
- 패스 2 : 80KB
- 심벌 테이블 : 20KB
- 공통 루틴 : 30KB
- 중첩 구동기(오버레이 드라이버) : 10KB

(a) 2단계 어셈블러 예

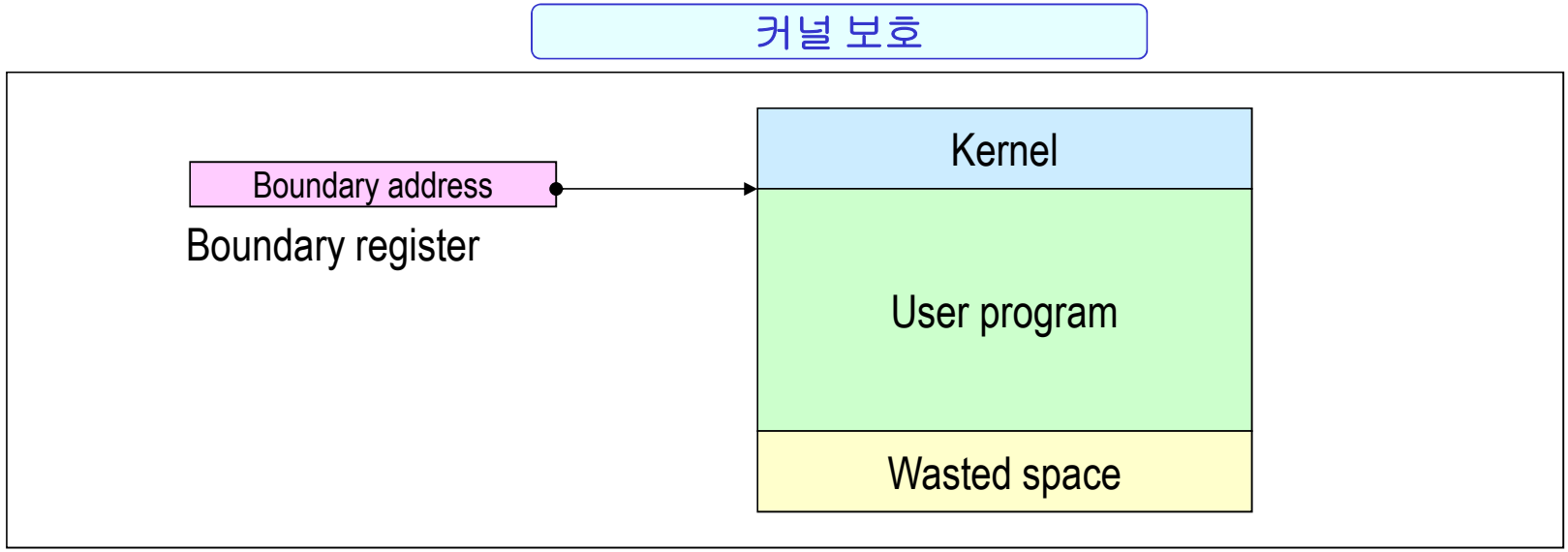


- 중첩 A : 심벌 테이블, 공통 루틴, 패스 1
- 중첩 B : 심벌 테이블, 공통 루틴, 패스 2

(b) 2단계 어셈블러 중첩

단일 분할 할당 기법(단일프로그래밍)

- 단일 분할 할당 기법 환경에서의 문제점(2)
  - 사용자 프로세스로부터 커널을 보호하는 기법 필요
    - 해결
      - 경계 레지스터 (boundary register) 사용

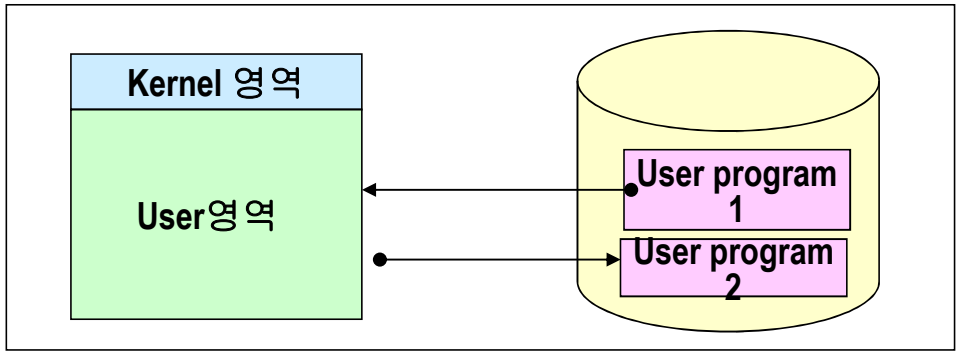


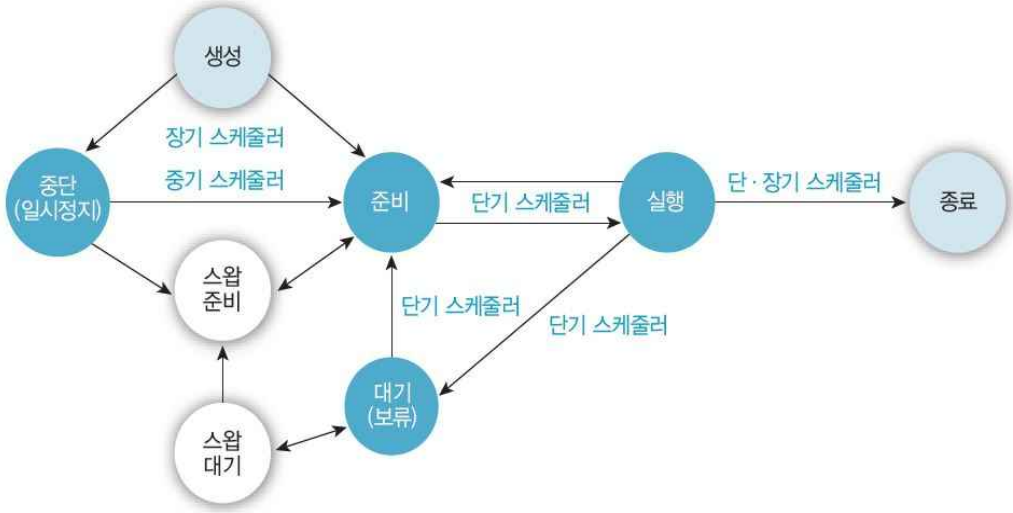
단일 분할 할당 기법(단일프로그래밍)

- 단일 분할 할당 기법 환경에서의 문제점**(3)**
  - 시스템 자원의 낭비
  - 시스템 성능의 저하
    - 해결
      - 다중 프로그래밍 기법 사용
      - 동시에 여러 프로그램들 적재되도록 함

단일 분할 할당 기법(단일프로그래밍)

- 스와핑(Swapping)기법
  - 하나의 프로그램 전체를 주기억장치에 할당하여 사용하다 필요에 따라 다른 프로그램과 교체하는 기법.
  - 실행이 종료되지 않은 프로세스가 할당받은 기억장소를 중간에 반납받을 수 있도록 하는 기법
    - 교체 기법을 사용하면 사용자 프로그램은 수행이 완료될 때까지 주기억장치에 적재될 필요가 없다.
    - 하나의 사용자 프로그램이 완료될 때까지 교체 과정을 여러 번 수행 할 수 있다.
    - 주기억장치에 있는 프로그램이 보조기억장치로 이동되는 것을 swap out, 보조기억장치에 있는 프로그램이 주기억장치로 이동되는 것을 swap in
    - 가상 기억장치의 페이징 기법으로 발전되었다.





세스의 스와핑 과정

기억장치 관리의 발전

기억장치 관리의 발전

연속 적재 기법 (Contiguous Loading)		분산 적재 기법 (Scatter Loading)						
실기억 공간 (Real Memory)							가상 기억 공간 (Virtual Memory)	
단일 사용자 (Single User)		다중 프로그래밍 (Multi-Programming)						
오버레이 Overlay	교체 Swapping	고정 분할 Fixed Partition	동적 분할 Dynamic Partition	페이징 Paging	세그먼테이션 Segmentation	페이지/세그먼트의 혼합 형태	요구 페이징	요구 세그먼테이션

오버레이 (overlay)기법
=> 프로그램 크기가 클때
스와핑(Swapping)기법
⇒시스템 성능 향상
⇒다중 프로그래밍 지원
⇒페이징 기법으로 발전

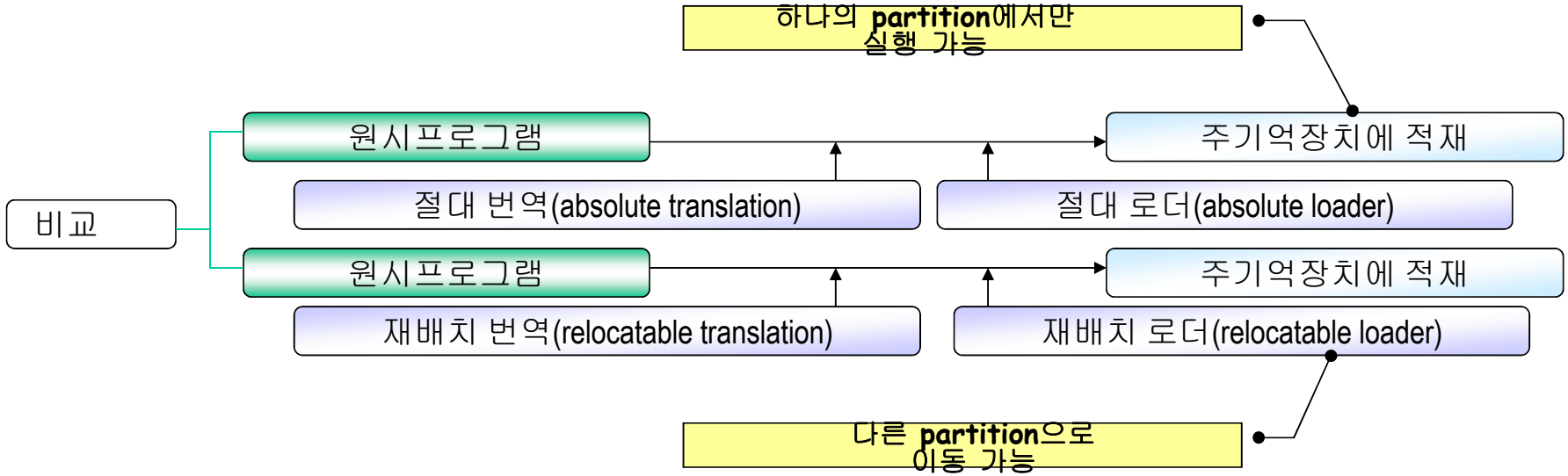
고정 분할 할당 기법(다중프로그래밍)

- 고정분할할당(Multiple contiguous Fixed parTition allocation, MFT)기법 = 정적할당 (static allocation)기법
  - 고정 분할 할당은 프로그램을 할당하기 전에 운영체제가 주기억 장치의 사용자 영역을 여러 개의 고정된 크기로 분할 하고 준비상태 큐에서 준비중인 프로그램을 각 영역에 할당하여 수행하는 기법이다.
  - 주기억장치를 일정 수의 고정된 크기들로 분할하여 실행 중인 여러 프로세스에게 할당
  - 주기억장치를 효율적으로 나누어 사용
  - 다중 프로그래밍을 위하여 여러 개의 작업이 동시에 컴퓨터의 주기억장치 내에 존재 가능
  - 프로그램을 실행하려면 프로그램 전체가 주기억장치에 위치해야 한다.
  - 프로그램이 분할된 영역보다 커서 영역 안에 들어갈 수 없는 경우가 발생할 수 있다.
  - 실행할 프로그램의 크기를 미리 알고 있어야 한다.
  - 현재는 사용되지 않는다.

영역 번호	영역 크기
1	5K
2	14K
3	10K
4	12K
5	16K

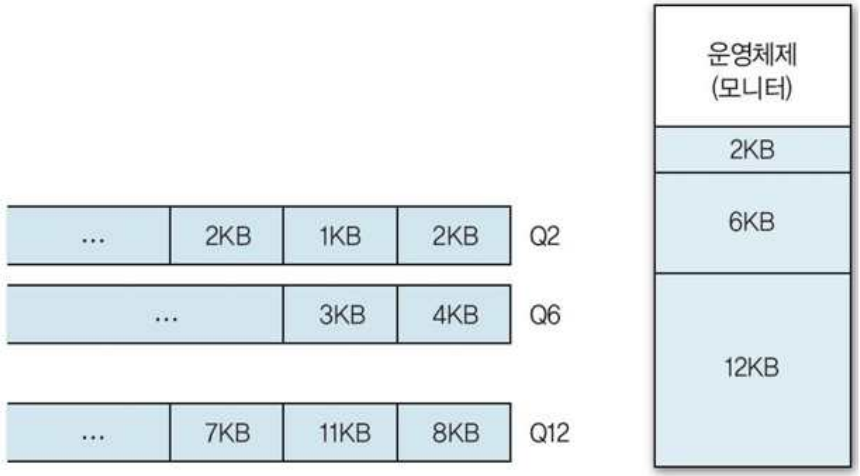
고정 분할 할당 기법(다중프로그래밍)

- 절대 번역 및 로딩(적재)
  - 절대어셈블러와 컴파일러에 의해 번역되며 정해진 영역 내에서만 실행된다. 분할마다 별도의 작업큐를 사용한다. 단점은 번역과정에서 결정 되므로 자신에게 할당된 분할에서만 사용
- 재배치 가능 번역 및 로딩(적재)
  - 절대로더의 단점 보완으로 모든 분할들에 대하여 하나의 작업큐를 사용하며 번역시 결정하는 것이 아니라 프로그램이 수행될 시점에 결정한다. 절대번역과 적재보다 기억장치의 관리가 효율적이지만, 번역기와 로더는 보다 복잡하다.

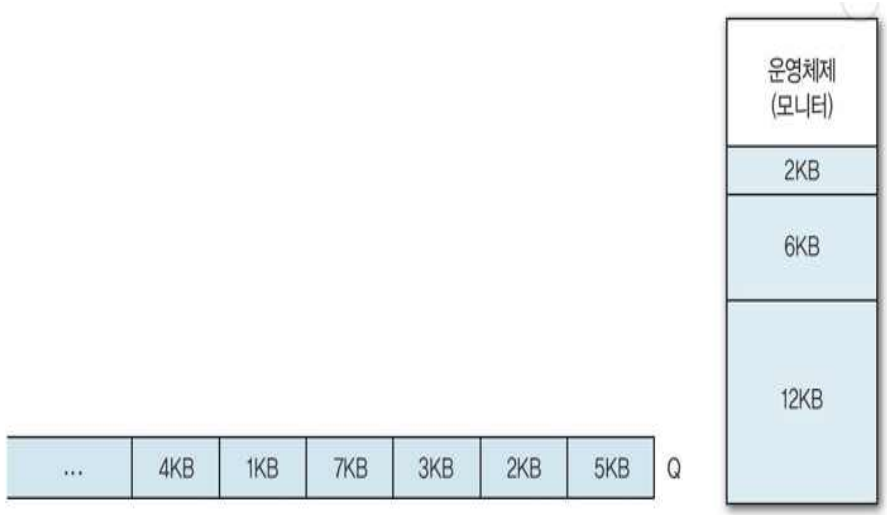


고정 분할 할당 기법(다중프로그래밍)

- 절대 번역 및 로딩(적재)
  - 절대어셈블러와 컴파일러에 의해 번역되며 정해진 영역 내에서만 실행된다. 분할마다 별도의 작업큐를 사용한다. 단점은 번역과정에서 결정 되므로 자신에게 할당된 분할에서만 사용
- 재배치 가능 번역 및 로딩(적재)
  - 절대로더의 단점 보완으로 모든 분할들에 대하여 하나의 작업큐를 사용하며 번역시 결정하는 것이 아니라 프로그램이 수행될 시점에 결정한다. 절대번역과 적재보다 기억장치의 관리가 효율적이지만, 번역기와 로더는 보다 복잡하다.



각 영역별로 독립된 큐가 있는 고정 분할 시스템



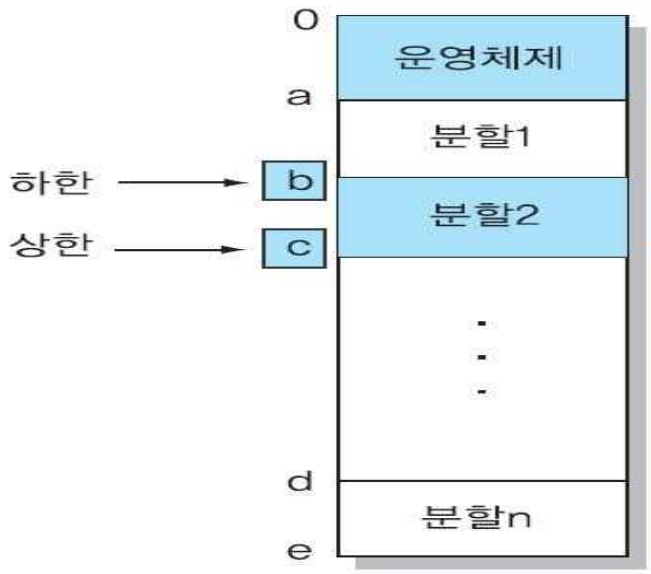
통합된 대기 큐가 있는 고정 분할 시스템

고정 분할 할당 기법(다중프로그래밍)

- 고정분할할당 기법의 문제점
  - 사용자 프로그램의 크기가 최대 분할 영역의 크기보다 큰 경우
    - 분할 영역 별로 중첩 구조를 사용하여 해결 가능
  - 커널과 다른 프로세스들에게 할당된 분할 영역들에 대한 보호 필요
    - 여러 개의 경계 레지스터를 사용하여 해결 가능
  - 각 분할 영역마다 낭비되는 공간 발생
    - 단편화(fragmentation)
      - 공간이 낭비되는 현상
    - 내부 단편화 (internal fragmentation)
      - 분할 영역 내에서 발생하는 공간의 낭비 현상
    - 외부 단편화 (external fragmentation)
      - 공간 용량의 문제로 한 분할 영역 전체가 낭비되는 현상

고정 분할 할당 기법(다중프로그래밍)

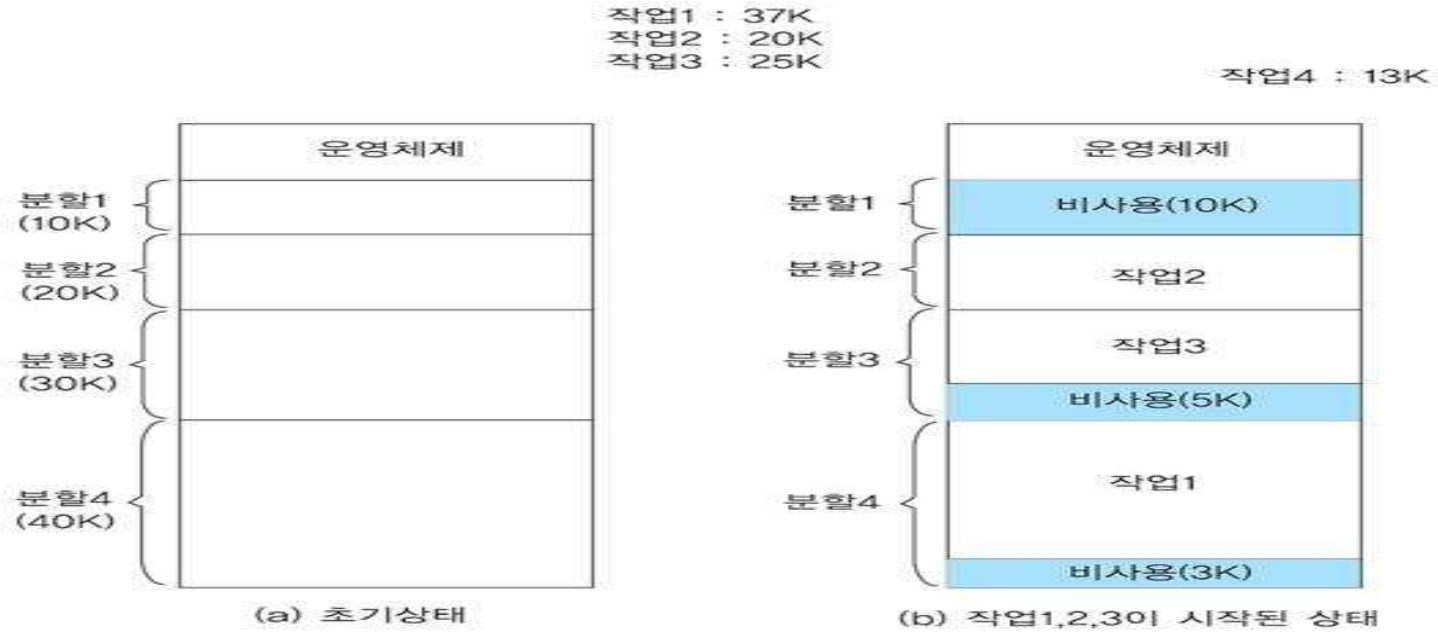
- 시스템 보호
  - 커널과 다른 프로세스들에게 할당된 분할 영역들에 대한 보호 필요
  - 주로 여러 개의 경계 레지스터**(boundary register)**를 이용



다중 프로그래밍 시스템에서의 프로그램 및 데이터 보호

고정 분할 할당 기법(다중프로그래밍)

단편화 : 각 분할 영역마다 낭비되는 공간 발생
: 내부 단편화, 외부 단편화



기억장치의 단편화 현상

고정 분할 할당 기법(다중프로그래밍)

[단편화 문제]

다음 표는 고정분할에서의 기억장치 단편화 현상을 보이고 있다. 내부 단편화 (Internal Fragmentation)는 얼마인가?

가. 480K **나. 430K**

다. 260K **라. 170K**

영역	분할의 크기	작업의 크기
A	20K	10K
B	50K	60K
C	120K	160K
D	200K	100K
E	300K	150K

학습 내용 정리

- 주기억장치 할당 기법
  - 단일 분할 할당 기법
    - 항상 시스템 내에 하나의 프로세스만 존재
    - 문제점 해결
    - 프로그램의 크기가 클 대: 오버레이 **(overlay)**기법
    - 프로그램 교체 :스와핑**(Swapping)**기법
    - 시스템 보호: 경계 레지스터 사용
    - 시스템 자원의 낭비, 시스템 성능 저하 : 다중프로그래밍 기법 사용
  - 고정 분할 할당 기법
    - 기억장치를 여러 개의 고정된 크키로 분할하는 기법
    - 정적 분할 다중 프로그래밍
    - 절대 로더와 재배치 로더에서 사용
    - 문제점
      - 시스템 보호 : 여러 개의 경계 레지스터를 사용하여 해결
      - 단편화 발생 : 내부 단편화, 외부 단편화 => 해결 방안 ?