인덕대학교

메모리 관리
- 주소 바인딩 및 할당

컴퓨터소프트웨어학과

김병국 교수

학습목표

- 논리적 주소와 물리적 주소의 차이를 안다.
- 시점에 따른 주소 바인딩 기술을 분류할 수 있다.
- 스와핑에 대한 개념을 안다.
- 프로세스 로딩과 메모리 할당 기법에 대해 이해한다.

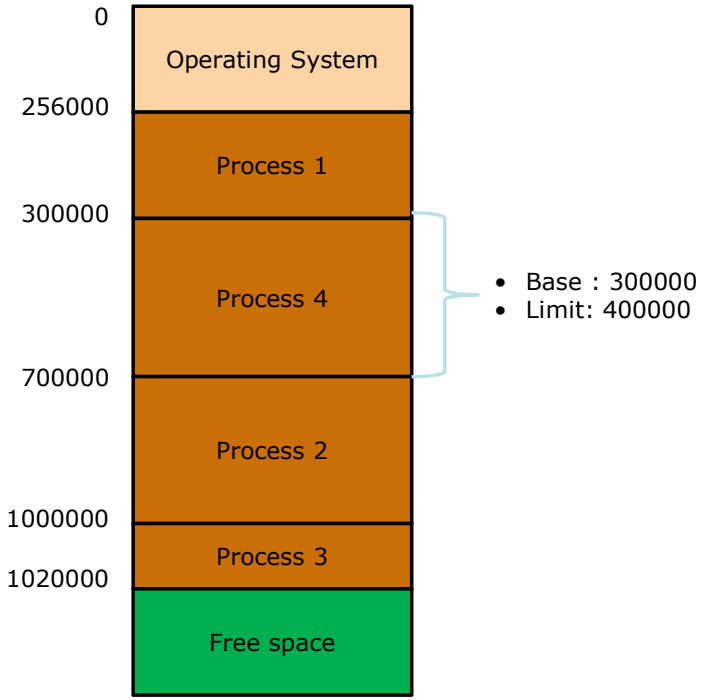
목차

- 메모리 주소 관리
- 주소 바인딩
- 스와핑
- 로딩 및 할당

1. 메모리 주소 관리 (1/2)

❑ 메모리 공간

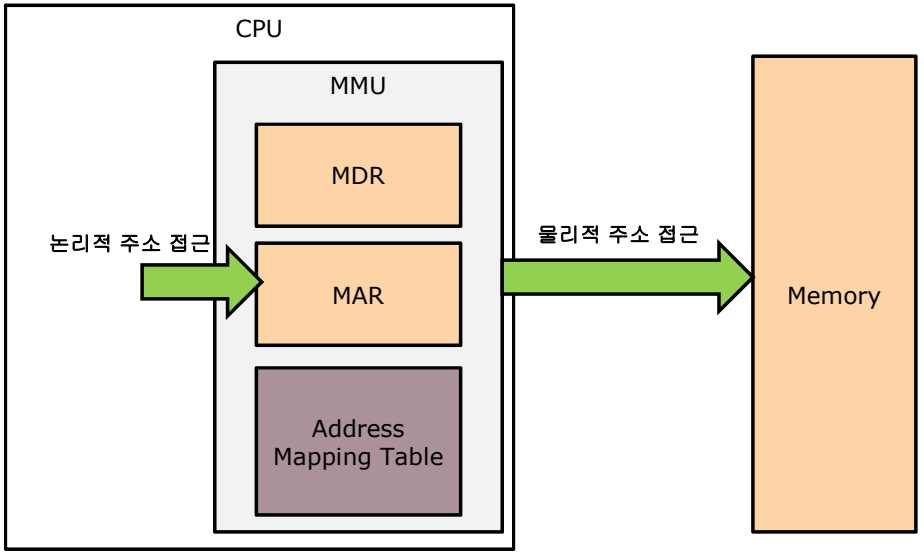
- 운영체제내 각 프로세스는 분리된 메모리 공간을 가짐
- PCB(Process Control Block)는 base 주소와 limit값을 포함
- 다른 프로세스 공간의 접근(침입)을 차단 ← 보안성 제공



1. 메모리 주소 관리 (2/2)

❑ 논리적 주소와 물리적 주소

- 논리적 주소(또는 가상 주소)
  - 다중 프로세스들의 공간을 쉽게 사용하기 위해 논리적 주소를 사용
  - 프로세스들은 논리적 주소를 사용(접근)
  - CPU가 취급하는 주소
  - 운영체제는 물리적 또는 논리적 주소를 접근하기 위해서 MMU(Memory Management Unit)를 제어
    - 논리적 주소에 대한 물리적 주소의 변환은 MMU에 의해 처리

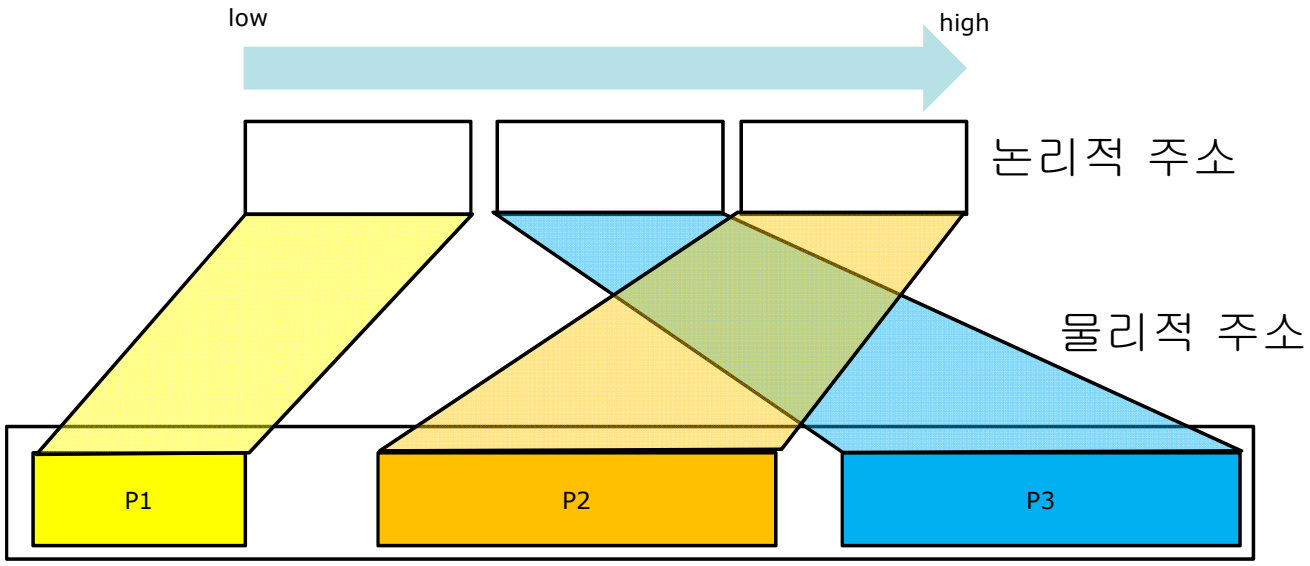


- 물리적 주소
  - 프로그램이 메모리에 저장되어있는 실제적인 공간
  - 메모리가 취급하는 주소
  - MMU의 MAR(Memory Address Register)에 의해 접근

2. 주소 바인딩 (1/4)

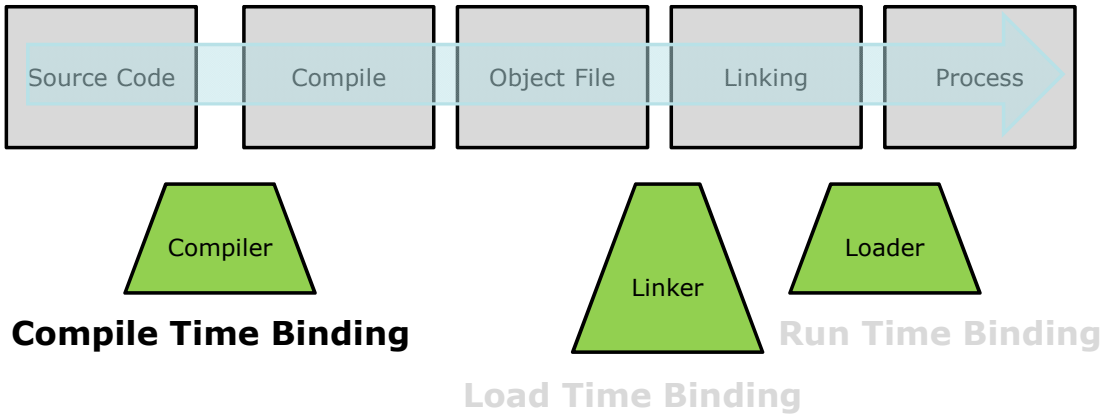
❑ 주소 바인딩(Address Binding) (1/4)

- 메모리의 물리적 주소(Physical Address)를 논리적(Logical) 주소와 연결(또는 매핑)하는 작업
- 시점에 따른 분류:
  - Compile Time Binding
  - Load Time Binding
  - Run Time Binding



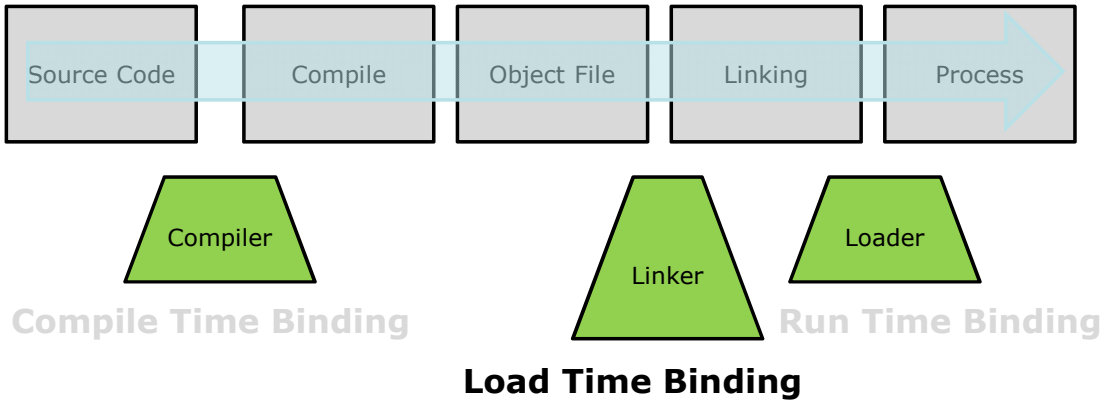
❑ 주소 바인딩(Address Binding) (2/4)

- Compile Time Binding
  - 프로그램내 변수 등의 기록될 영역에 대하여 재할당(relocatable) 형태의 주소로 설정
    - 예: 일반 변수 등
  - 성공적인 구동을 위해서는 프로그램 전체가 메모리에 적재 되어야 함
  - 심볼릭(symbolic) 주소 형태로 바인딩 처리
- Load Time Binding
- Run Time Binding



❑ 주소 바인딩(Address Binding) (3/4)

- Compile Time Binding
- Load Time Binding
  - 메모리의 적재될 위치는 알지 못하지만, 라이브러리들에 대한 위치를 **상대 주소**로 할당
  - 라이브러리들이 적재될 때 시스템에 의해 위치가 재설정
  - 성공적인 구동을 위해서는 프로그램 전체가 메모리에 적재 되어야 함
- Run Time Binding

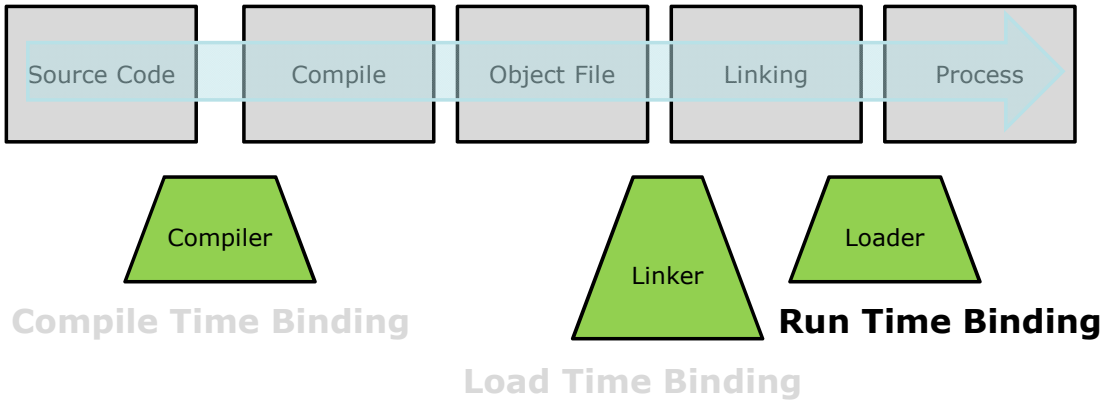


2. 주소 바인딩 (4/4)

인덕대학교

❑ 주소 바인딩(Address Binding) (3/4)

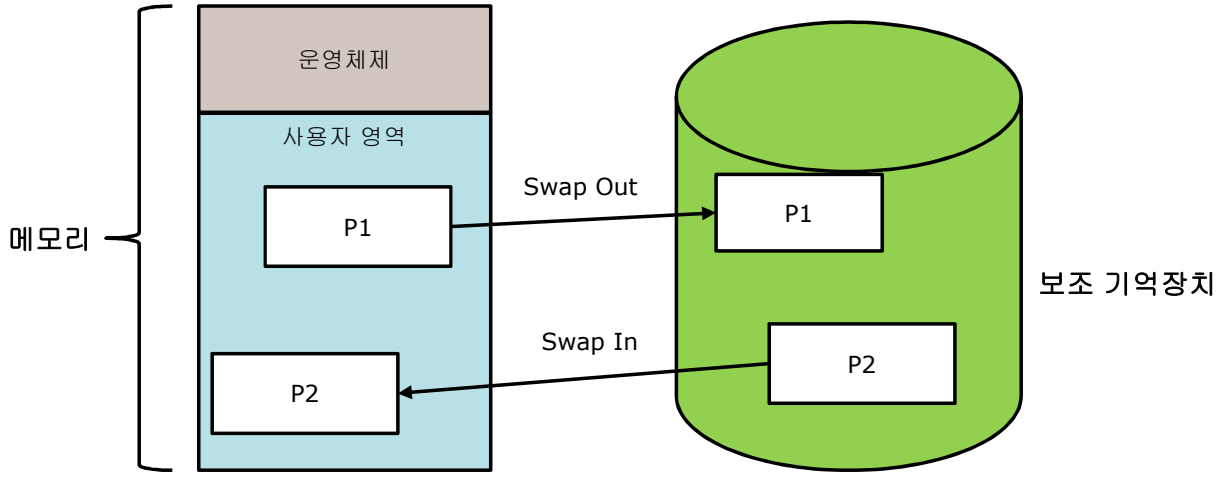
- Compile Time Binding
- Load Time Binding
- Run Time Binding
  - 프로세스가 running 상태일 때, 위치가 새로 할당되는 구조
  - 스케줄러에 의해 프로세스들의 메모리내 위치가 변경
  - 대부분의 운영체제가 그러함
  - CPU는 지정된 주소로 접근을 시도하지만, **MMU에 의해 새로 변경된 곳으로 접근됨**



3. 스와핑 (1/3)

❑ 스와핑(Swapping)

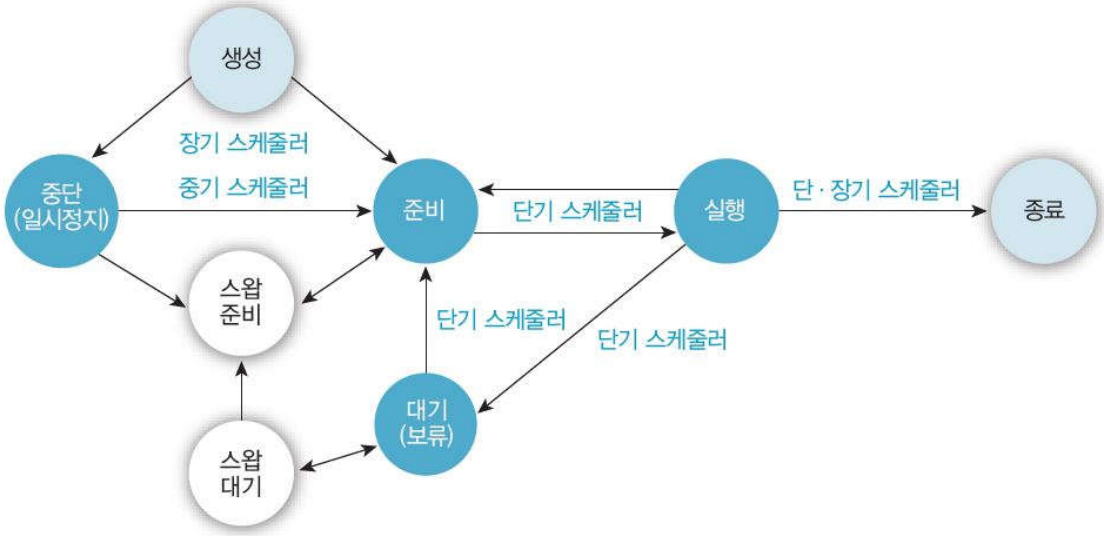
- 보류(suspend) 상태의 프로세스에 대하여 임시로 보조 기억장치에 백업
- 메모리 절약 기법
- 기능에 따른 분류:
  - 스왑 아웃(Swap Out)
  - 스왑 인(Swap In)



3. 스와핑 (2/3)

❑ 스와핑(Swapping)

- 프로세스의 상태에 따라 일부 프로세스는 보조 기억장치에 백업 및 복구
- 스와핑 작업이 너무 자주 발생하면 I/O접근에 따른 지연으로 인해 전반적인 성능이 저하 발생
- 해결안:
  - 충분한 공간의 메모리를 탑재
  - 프로세스의 수를 감소



3. 스와핑 (3/3)

❑ 고려사항

- Swap-In 방법
  - 동일한 주소로 복귀(Binding 문제가 간단해 짐)
  - Execution Time Binding의 경우 메모리 어느 곳이든 가져올 수 있음

- Context-switch time
  - 주로 transfer time에 많은 시간이 소요
    - 시간은 swap되는 양에 비례
  - 프로세스에 할당되는 타임 슬롯은 Swapping 시간보다 충분히 길어야 함

4. 로딩 및 할당 (1/8)

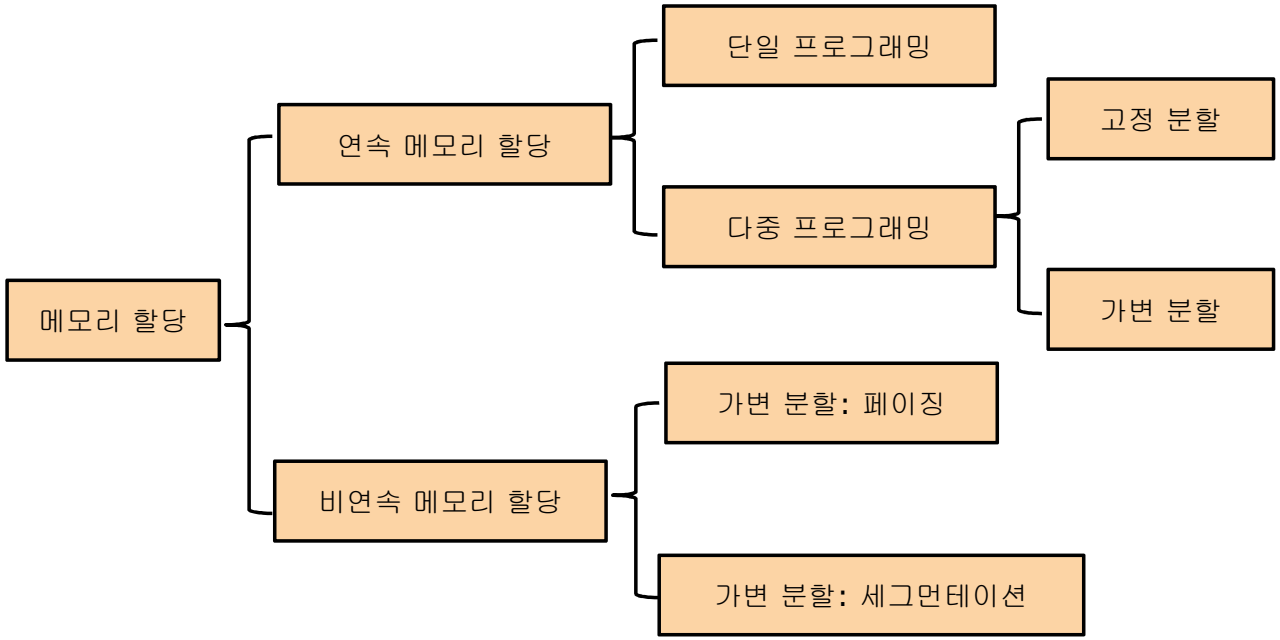
❑동적 로딩(Dynamic Loading)

- 모든 루틴(routine)을 교체 가능한 형태로 저장
- 해당 루틴이 호출되기 전까지는 메모리에 적재하지 않음
  - Main() 루틴과 기본 정적 함수 부분만 처음에 적재
  - 동적 함수 호출 시 그 때 Address Binding하는 구조
- 메모리의 공간을 효율적(절약)으로 사용할 수 있음
- 동적 라이브러리 개념

4. 로딩 및 할당 (2/8)

❑ 할당(Allocation)

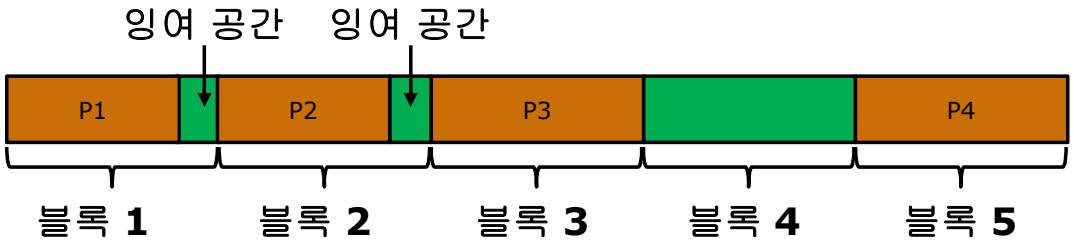
- 연속 메모리 할당 방법(Contiguous Allocation)
- 비연속(분산) 메모리 할당 방법(Partition Allocation)



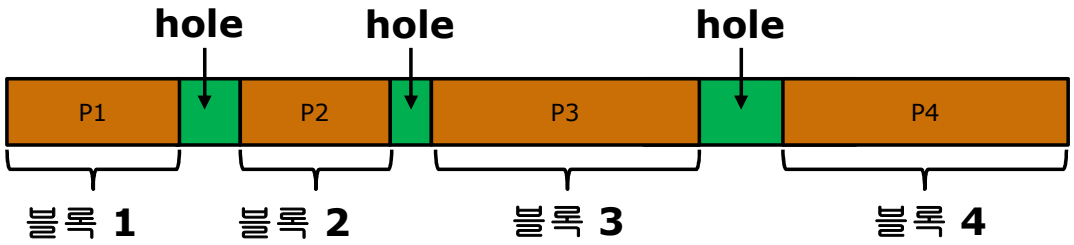
4. 로딩 및 할당 (3/8)

❑ 단편화(Fragmentation)

- Internal Fragmentation
  - 자원을 **고정된 블록**으로 나누어 사용하게 될 경우
  - 블록보다 작은 크기의 내용물이 기록될 때, 잉여 공간(free space)생기게 됨



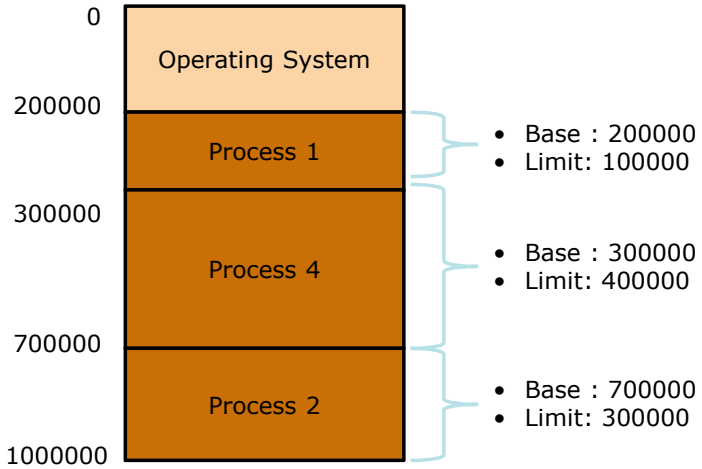
- External Fragmentation
  - 가변 크기를 갖는 블록들을 동적으로 할당하여 운영
  - 블록 간의 공간(hole)이 생기게 됨



4. 로딩 및 할당 (3/8)

❑ 연속 메모리 할당(Contiguous Allocation) (1/6)

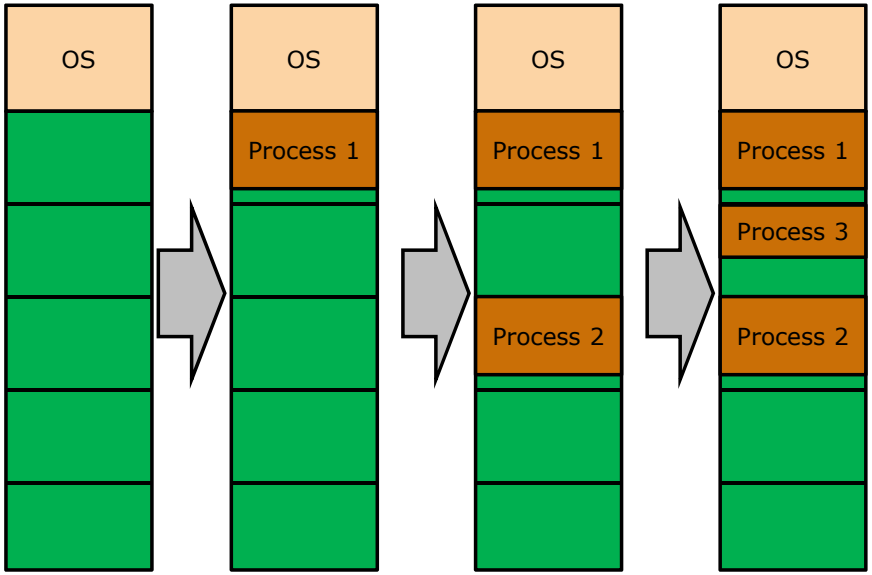
- 주 기억장치는 OS와 사용자 프로세스들에게 공간을 제공해야함
- 주 기억장치는 한정적이기 때문에 효율적으로 프로세스들에게 할당되어야 함
- **연속 메모리 할당 기법은 초창기 기법 중 하나임**
- 주 기억장치는 일반적으로 두 영역(partition)으로 구분
  - OS : 일반적으로 낮은 주소 영역에 할당
  - 사용자 프로세스 : 운영체제가 사용하지 않는 높은 주소 영역에 할당



4. 로딩 및 할당 (4/8)

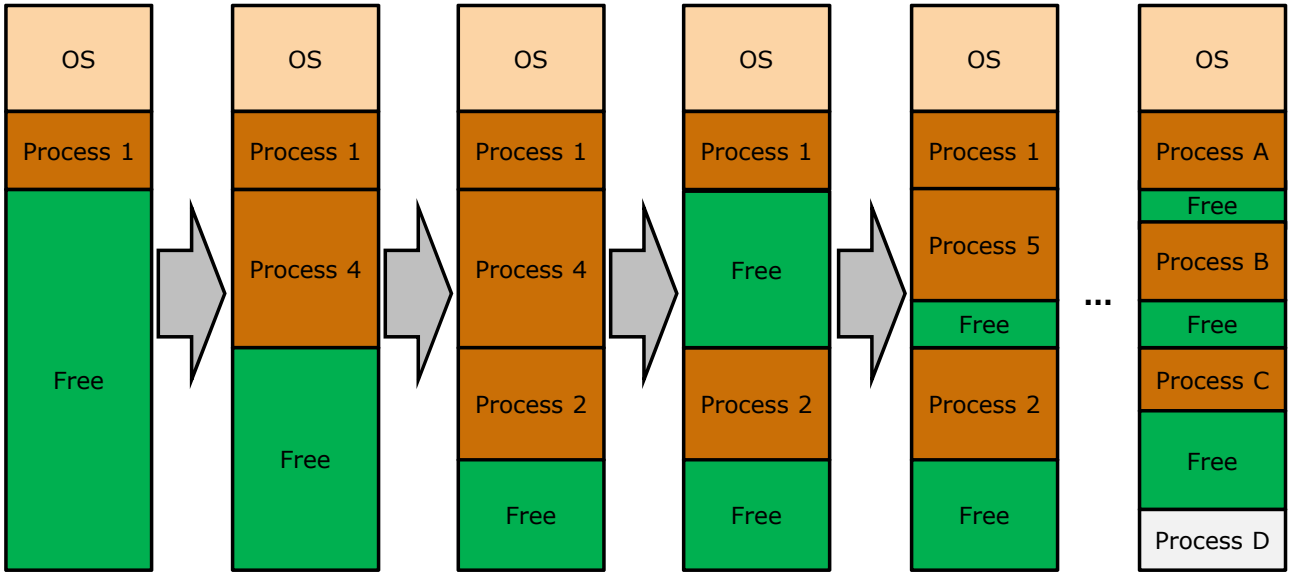
❑ 연속 메모리 할당(Contiguous Allocation) (2/6)

- 고정 분할(Fixed Partitions)
  - 프로세스를 위한 고정된 크기의 파티션으로 메모리 공간을 할당
  - 장점: 메모리 관리 단순
  - 단점
    - 메모리 낭비 발생
    - Internal Fragmentation 발생
    - 파티션보다 큰 프로세스에 대해서는 할당 불가



❑연속 메모리 할당(Contiguous Allocation) (3/6)

- 가변 분할 (1/3)
  - 프로세스의 생성과 소멸의 잦은 발생 시 빈 공간이 발생 → HOLE
  - External Fragmentation 발생

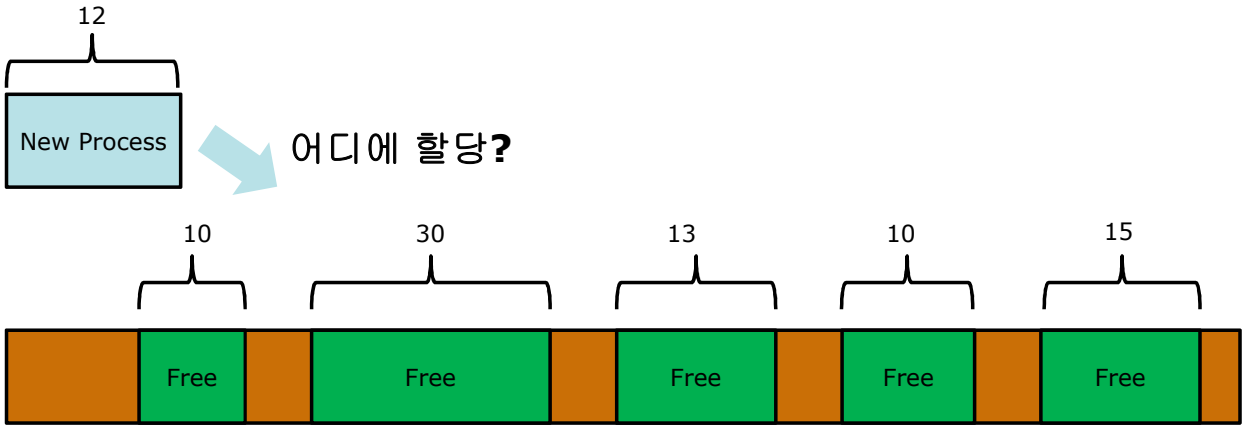


4. 로딩 및 할당 (6/8)

❑ 연속 메모리 할당(Contiguous Allocation) (4/6)

- 가변 분할 (2/3)
  - 분산 holes에 대하여 새로운 프로세스가 생성될 때 점유하는 방식
  - Holes에 대하여 배열(Array) 또는 연결리스트(Linked List)로 관리

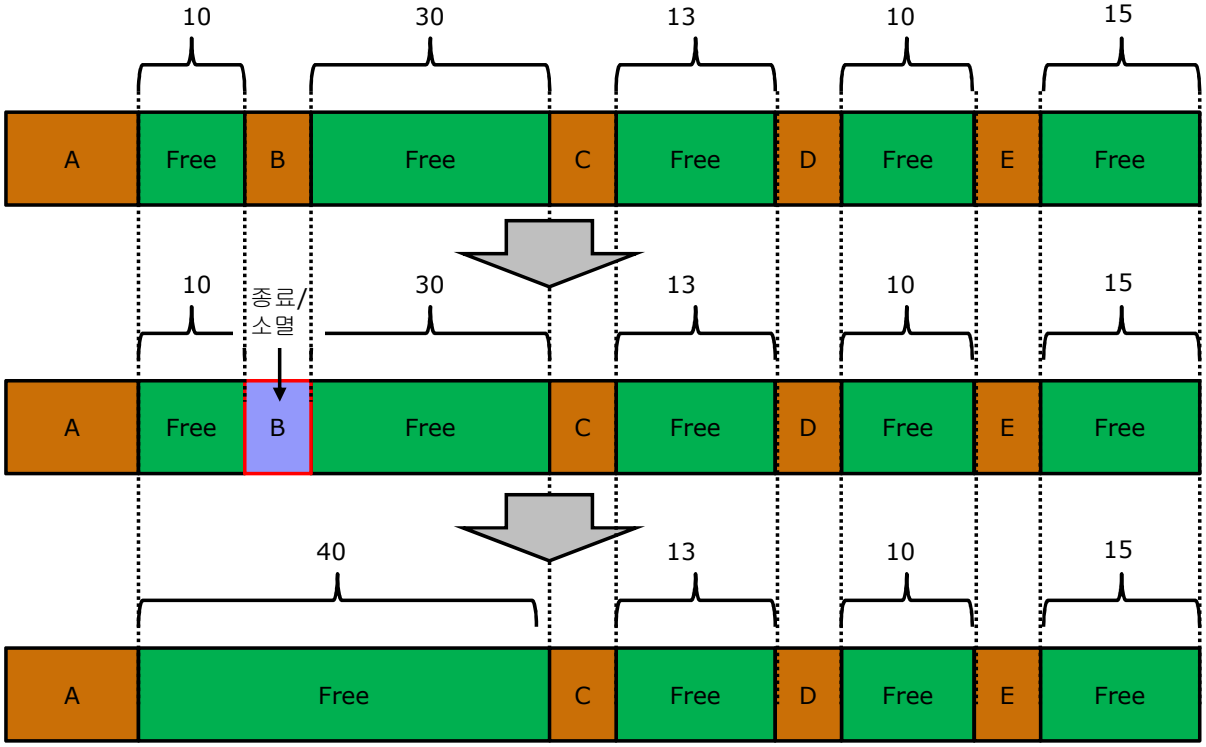
- 선택 기법:
  - **First-Fit**: 프로세스가 들어갈 수 있는 첫번째 hole를 선택
  - **Best-Fit**: 점유할 때 가장 작은 hole이 발생될 수 있는 공간을 선택
  - **Worst-Fit**: 가장 큰 hole을 선택



4. 로딩 및 할당 (7/8)

❑ 연속 메모리 할당(Contiguous Allocation) (5/6)

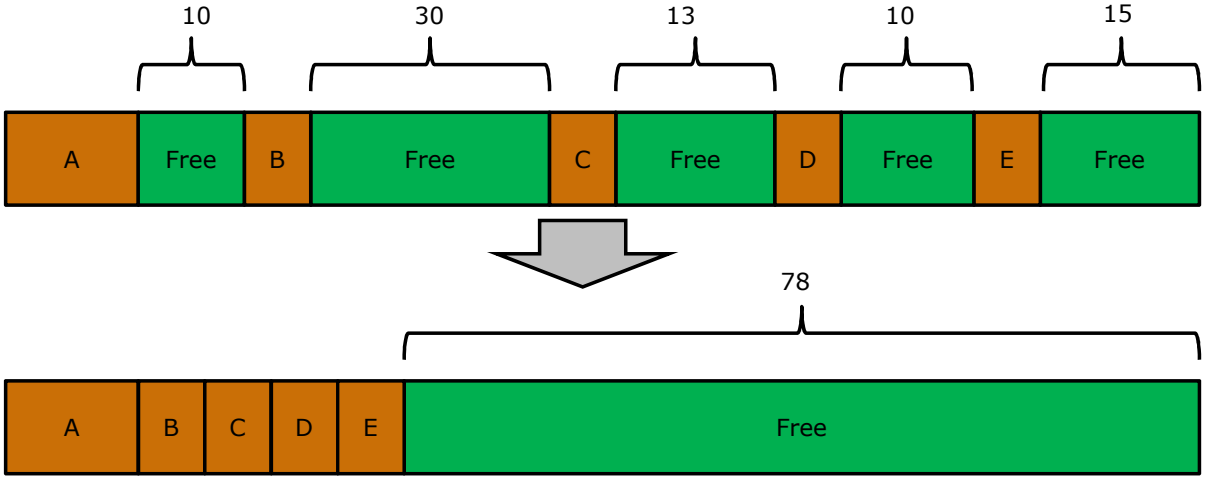
- 메모리 통합
  - 프로세스의 종료로 인해 반환된 hole들에 대한 통합이 필요
  - Hole들이 인접해 있을 때, 하나의 홀로 통합
  - 충분한 크기의 hole를 확보하여 다른 프로세스의 생성이 가능하도록 자원을 관리하는 목적



4. 로딩 및 할당 (8/8)

❑ 연속 메모리 할당(Contiguous Allocation) (6/6)

- 메모리 압축
  - 분산된 프로세스 영역을 이동
  - **충분한 크기의 hole를 확보**하여 다른 프로세스의 생성이 가능하도록 자원을 관리하는 목적
  - 단점:
    - 운영체제의 잦은 메모리 최적화작업에 따른 **성능 저하 발생**
    - 실시간 시스템의 경우 **대기시간이 길어짐**



수고하셨습니다.