Chapter 07

물리 메모리 관리

Contents

학습목표

- 메모리 관리의 복잡성과 이중성을 이해하고 메모리 관리자의 역할을 파악한다.
- 절대 주소와 상대 주소의 의미, 절대 주소의 상대 주소 변환 과정을 이해한다.
- 메모리 오버레이와 스왑 기법을 알아본다.
- 가변 분할 방식과 고정 분할 방식을 메모리 관리 방식을 알아본다.
- 버디 시스템의 동작을 이해한다.

1 메모리 관리의 복잡성

메모리 주소

- 1B로 나뉜 메모리의 각 영역은 메모리 주소로 구분하는데 보통 0번지부터 시작
- CPU는 메모리에 있는 내용을 가져오거나 작업 결과를 메모리에 저장하기 위해 메모리 주소 레지스터(MAR)를 사용

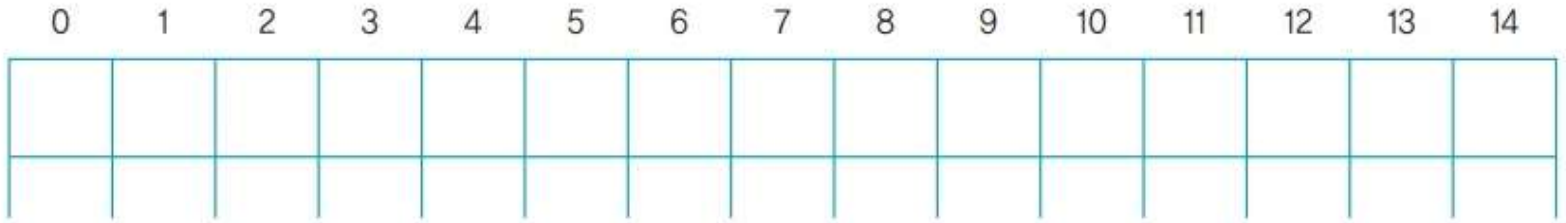
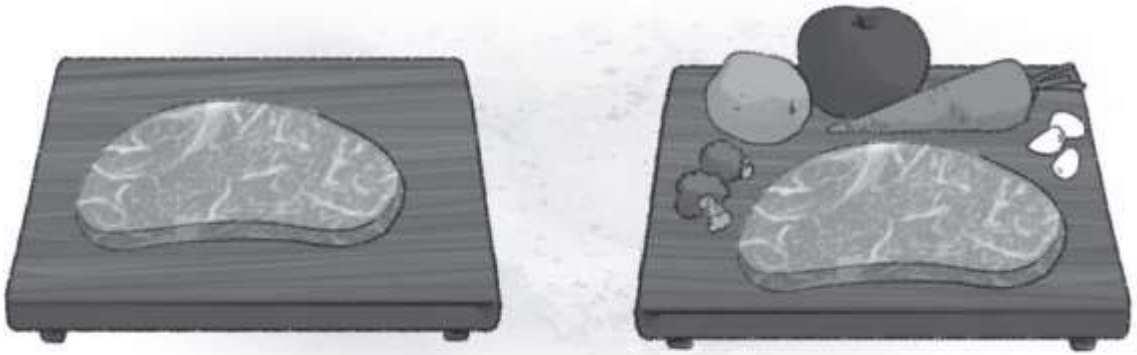


그림 7-1 메모리 주소

1 메모리 관리의 복잡성

메모리 관리의 복잡성

- 메모리는 폰노이만 구조의 컴퓨터에서 유일한 작업 공간이며 모든 프로그램은 메모리에 올라와야 실행 가능
- 시분할 시스템에서는 운영체제를 포함한 모든 응용 프로그램이 메모리에 올라와 실행되기 때문에 메모리 관리가 복잡



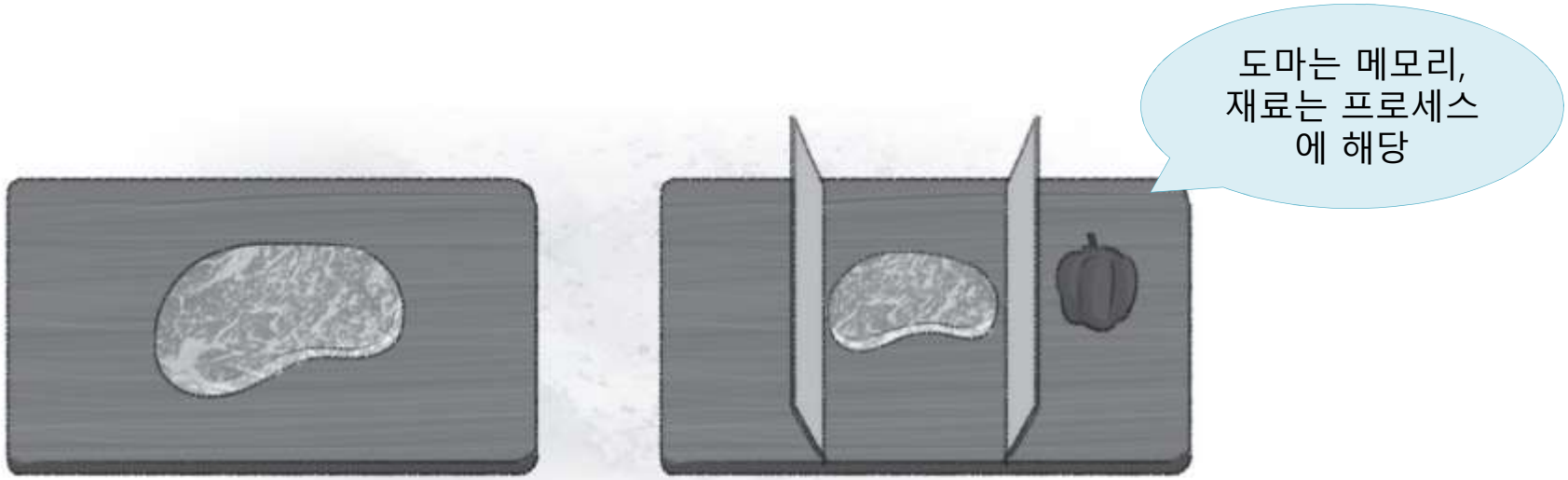
(a) 일괄 처리 시스템 (b) 시분할 시스템

그림 7-2 메모리 관리의 복잡성

2 메모리 관리의 이중성

메모리 관리의 이중성

- 프로세스 입장에서는 메모리를 독차지하려 하고,
  메모리 관리자 입장에서는 되도록 관리를 효율적으로 하고 싶어함



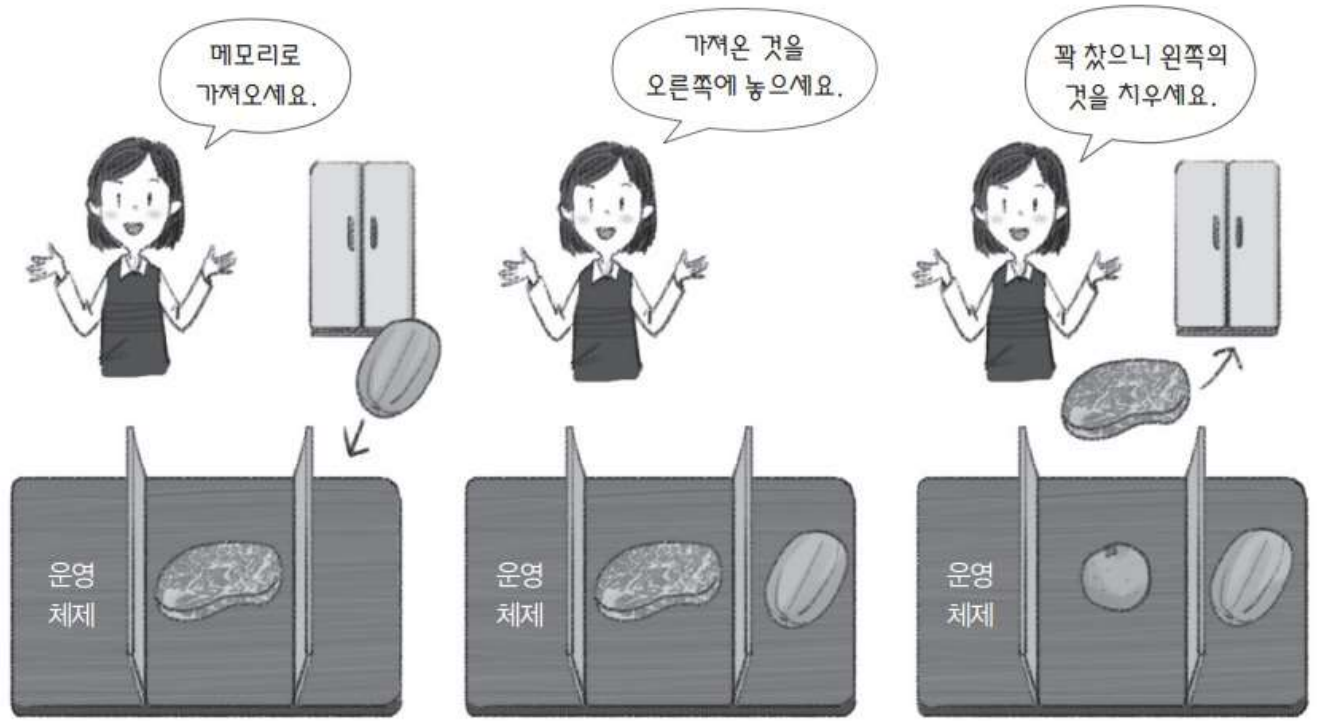
(a) 프로세스 입장의 메모리 (b) 메모리 관리자 입장의 메모리

그림 7-3 **메모리 관리의 이중성**

4 메모리 관리자의 역할

메모리 관리자Memory Manage Unit, MMU

- 메모리 관리를 담당하는 하드웨어



(a) 가져오기 (b) 배치 (c) 재배치

그림 7-7 메모리 관리자의 작업

4 메모리 관리자의 역할

메모리 관리자의 작업

- 가져오기 (fetch) 작업 : 프로세스와 데이터를 메모리로 가져옴
- 배치(placement) 작업 : 가져온 프로세스와 데이터를 메모리의 어떤 부분에 올려놓을지 결정
- 재배치(replacement) 작업 : 꽉 차 있는 메모리에 새로운 프로세스를 가져오기 위해 오래된 프로세스를 내보냄

메모리 관리자의 정책

- 가져오기 정책: 프로세스가 필요로 하는 데이터를 언제 메모리로 가져올지 결정하는 정책
- 배치 정책 : 가져온 프로세스를 메모리의 어떤 위치에 올려놓을지 결정하는 정책
- 재배치 정책: 메모리가 꽉 찼을 때 메모리 내에 있는 어떤 프로세스를 내보낼지 결정하는 정책

1 32bit CPU와 64bit CPU의 차이

CPU의 비트

- 한 번에 다룰 수 있는 데이터의 최대 크기
- 32bit CPU는 한 번에 다룰 수 있는 데이터의 최대 크기가 32bit
- 32bit CPU 내의 레지스터 크기는 전부 32bit, 산술 논리 연산장치와 대역폭도 32bit

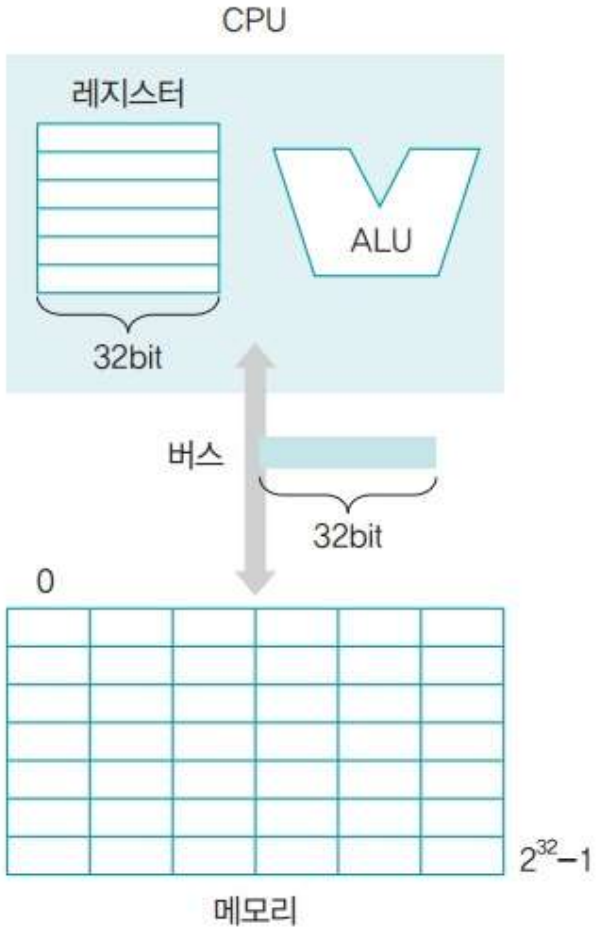


그림 7-8 32bit CPU의 구조

1 32bit CPU와 64bit CPU의 차이

32bit CPU의 메모리 크기

- 표현할 수 있는 메모리 주소의 범위가 0~2^{32}-1, 총개수가 2^{32}
- 16진수로 나타내면 00000000~FFFFFFFF, 총크기는 2^{32}B(약 4GB)

64bit CPU의 메모리 크기

- 0~2^{64}-1번지의 주소 공간을 제공
- 총 크기가 2^{64}B, 약 16,777,216TB로 거의 무한대에 가까운 메모리 사용 가능

표 7-2 컴퓨터의 메모리 크기

구분	32bit CPU	64bit CPU
주소 범위	0~2^{32}-1번지	0~2^{64}-1번지
총크기	2^{32}B(약 4GB)	2^{64}B(약 16,777,216TB)

물리 주소 공간과 논리 주소 공간

- 물리 주소 공간 : 하드웨어 입장에서 바라본 주소 공간으로 컴퓨터마다 크기가 다름
- 논리 주소 공간 : 사용자 입장에서 바라본 주소 공간

2 절대 주소와 상대 주소

단순 메모리 구조

- 한 번에 한 가지 일만 처리하는 일괄 처리 시스템에서 볼 수 있음
- 메모리를 운영체제 영역과 사용자 영역으로 나누어 관리

그림 7-9 단순 메모리 구조

2 절대 주소와 상대 주소

단순 메모리 구조에서 사용자 프로세스 적재

- 사용자 프로세스는 운영체제 영역을 피하여 메모리에 적재
- 사용자 프로세스가 운영체제의 크기에 따라 매번 적재되는 주소가 달라지는 것은 번거로움. 이를 개선하여 사용자 프로세스를 메모리의 최상위부터 사용

→ 그러나 메모리를 거꾸로 사용하기 위해 주소를 변경하는 일이 복잡하기 때문에 잘 쓰이지 않음

그림 7-10 상위 메모리부터 사용자 영역 할당

2 절대 주소와 상대 주소

경계 레지스터(base register)

- 운영체제 영역과 사용자 영역 경계 지점의 주소를 가진 레지스터
- CPU 내에 있는 경계 레지스터가 사용자 영역이 운영체제 영역으로 침범하는 것을 막아줌
- 메모리 관리자는 사용자가 작업을 요청할 때마다 경계 레지스터의 값을 벗어나는지 검사하고, 만약 경계 레지스터를 벗어나는 작업을 요청하는 프로세스가 있으면 그 프로세스를 종료

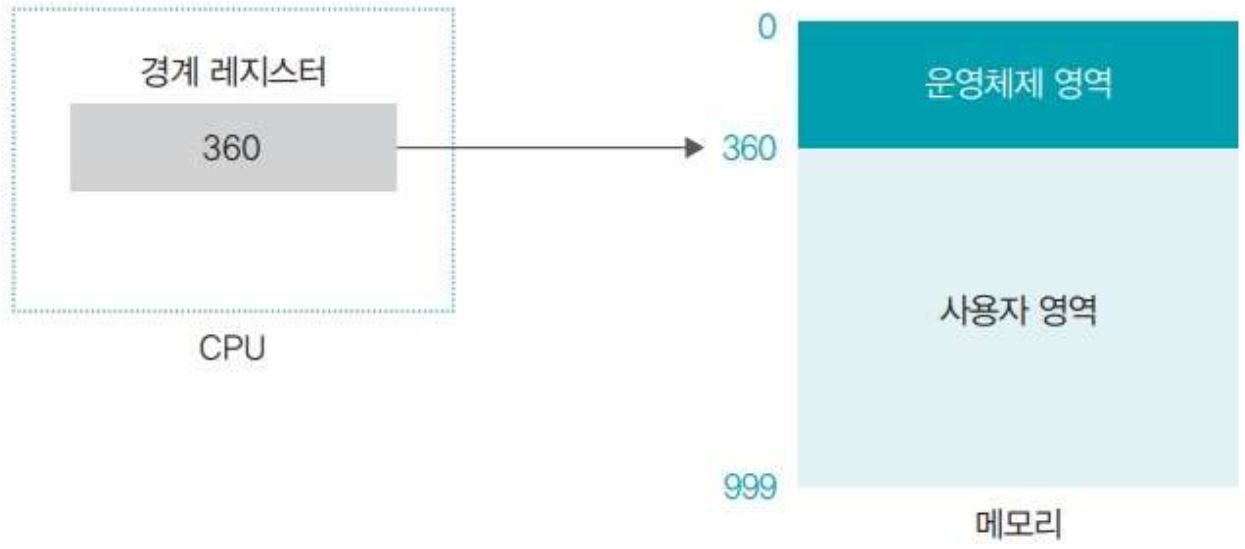


그림 7-11 운영체제와 경계 레지스터

2 절대 주소와 상대 주소

절대 주소absolute address

- 실제 물리 주소physical address를 가리키는 주소
- 메모리 주소 레지스터가 사용하는 주소
- 컴퓨터에 꽂힌 램 메모리의 실제 주소

상대 주소relative address

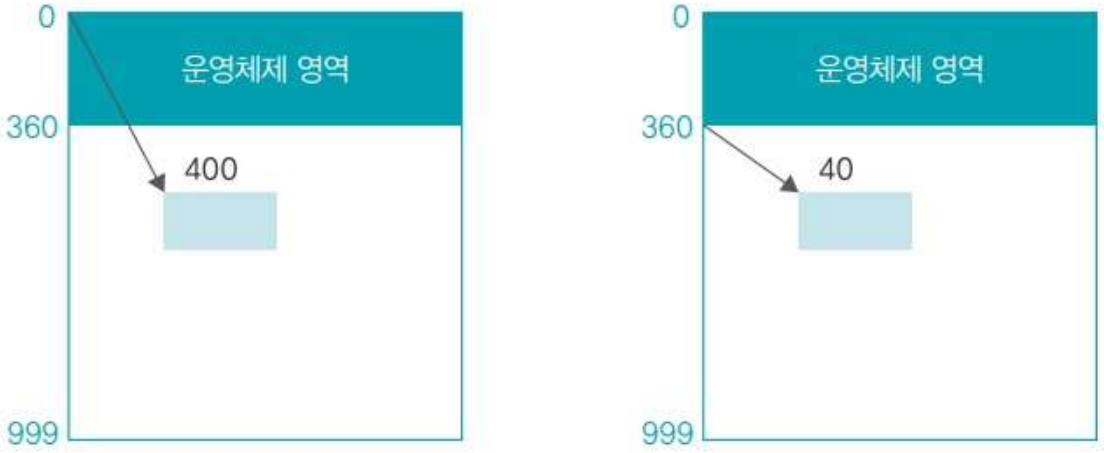
- 사용자 영역이 시작되는 번지를 0번지로 변경하여 사용하는 주소
- 사용자 프로세스 입장에서 바라본 주소
- 절대 주소와 관계없이 항상 0번지 부터 시작
- 프로세스 입장에서 상대 주소가 사용할 수 없는 영역의 위치를 알 필요가 없고, 주소가 항상 0번지부터 시작하기 때문에 편리

절대 주소와 상대 주소의 차이

- 논리 주소 공간은 상대 주소를 사용하는 주소 공간
- 물리 주소 공간은 절대 주소를 사용하는 주소 공간

2 절대 주소와 상대 주소

절대 주소와 상대 주소의 차이



(a) 절대 주소 (b) 상대 주소

그림 7-12 **절대 주소와 상대 주소**

표 7-3 **절대 주소와 상대 주소**

구분	절대 주소	상대 주소
관점	메모리 관리자 입장	사용자 프로세스 입장
주소 시작	물리 주소 0번지부터 시작	물리 주소와 관계없이 항상 0번지부터 시작
주소 공간	물리 주소(실제 주소) 공간	논리 주소 공간

2 절대 주소와 상대 주소

상대 주소를 절대 주소로 변환하는 과정

- 메모리 접근 시 상대 주소를 사용하면 절대 주소로 변환해야 함
- 메모리 관리자는 사용자 프로세스가 상대 주소를 사용하여 메모리에 접근할 때마다 상대 주소값에 재배치 레지스터 값을 더하여 절대 주소를 구함
- 재배치 레지스터는 주소 변환의 기본이 되는 주소값을 가진 레지스터로, 메모리에서 사용자 영역의 시작 주소값이 저장

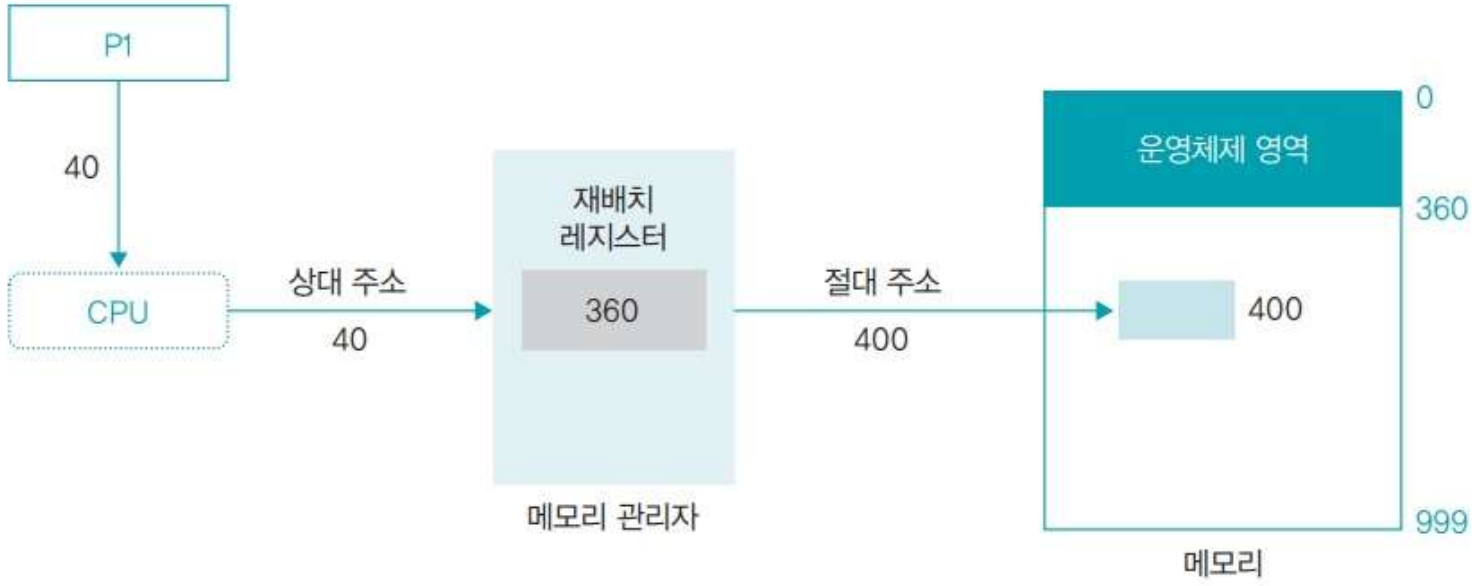


그림 7-13 상대 주소를 절대 주소로 변환하는 과정

1 메모리 오버레이

메모리 오버레이

- 프로그램의 크기가 실제 메모리(물리 메모리)보다 클 때 전체 프로그램을 메모리에 가져오는 대신 적당한 크기로 잘라서 가져오는 기법

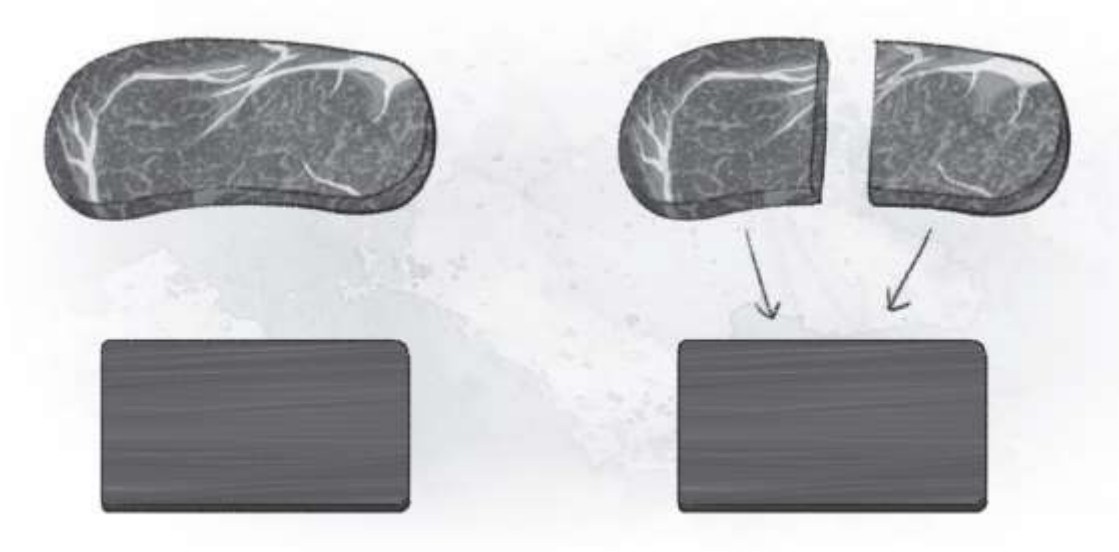


그림 7-14 도마보다 큰 고깃덩어리를 손질하는 방법

1 메모리 오버레이(overlay)

메모리 오버레이의 작동 방식

- 프로그램이 실행되면 필요한 모듈만 메모리에 올라와 실행

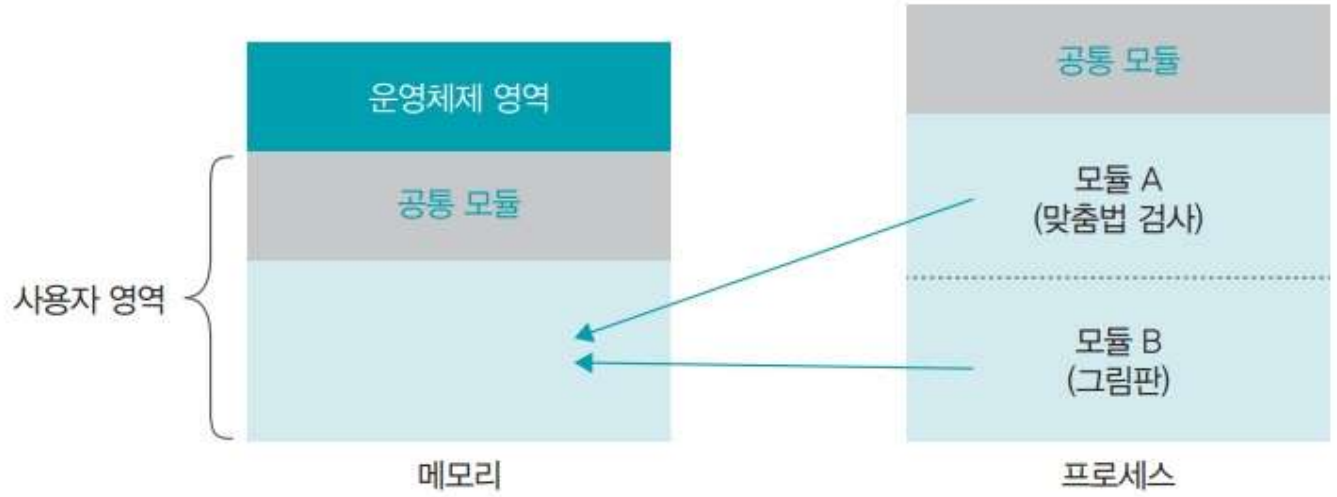


그림 7-15 메모리 오버레이의 작동 방식

메모리 오버레이의 의미

- 한정된 메모리에서 메모리보다 큰 프로그램 실행 가능
- 프로그램 전체가 아니라 일부만 메모리에 올라와도 실행 가능

2 스왑

스왑 영역swap area

- 메모리가 모자라서 쫓겨난 프로세스를 저장장치의 특별한 공간에 모아두는 영역
- 메모리에서 쫓겨났다가 다시 돌아가는 데이터가 머무는 곳이기 때문에 저장장치는 장소만 빌려주고 메모리 관리자가 관리
- 사용자는 실제 메모리의 크기와 스왑 영역의 크기를 합쳐서 전체 메모리로 인식하고 사용

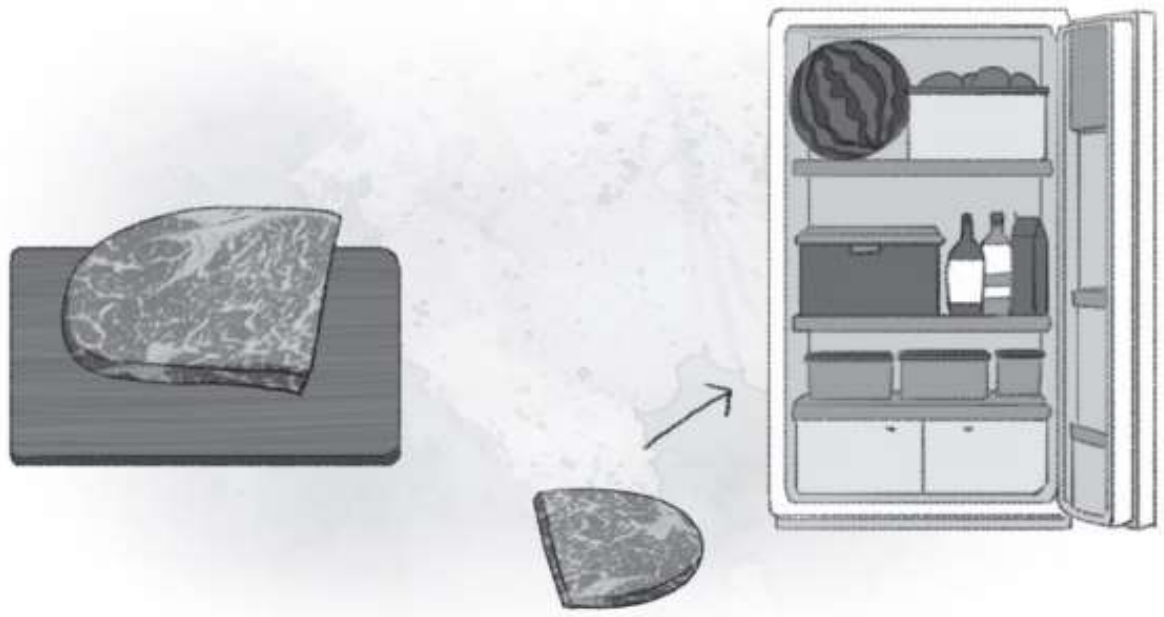


그림 7-16 다른 고깃덩어리를 손질하기 위해 고깃덩어리를 보관 창고에 저장하는 경우

2 스왑

스왑인 Swap-in과 스왑아웃 Swap-out

- 스왑인 : 스왑 영역에서 메모리로 데이터를 가져오는 작업
- 스왑아웃 : 메모리에서 스왑 영역으로 데이터를 내보내는 작업

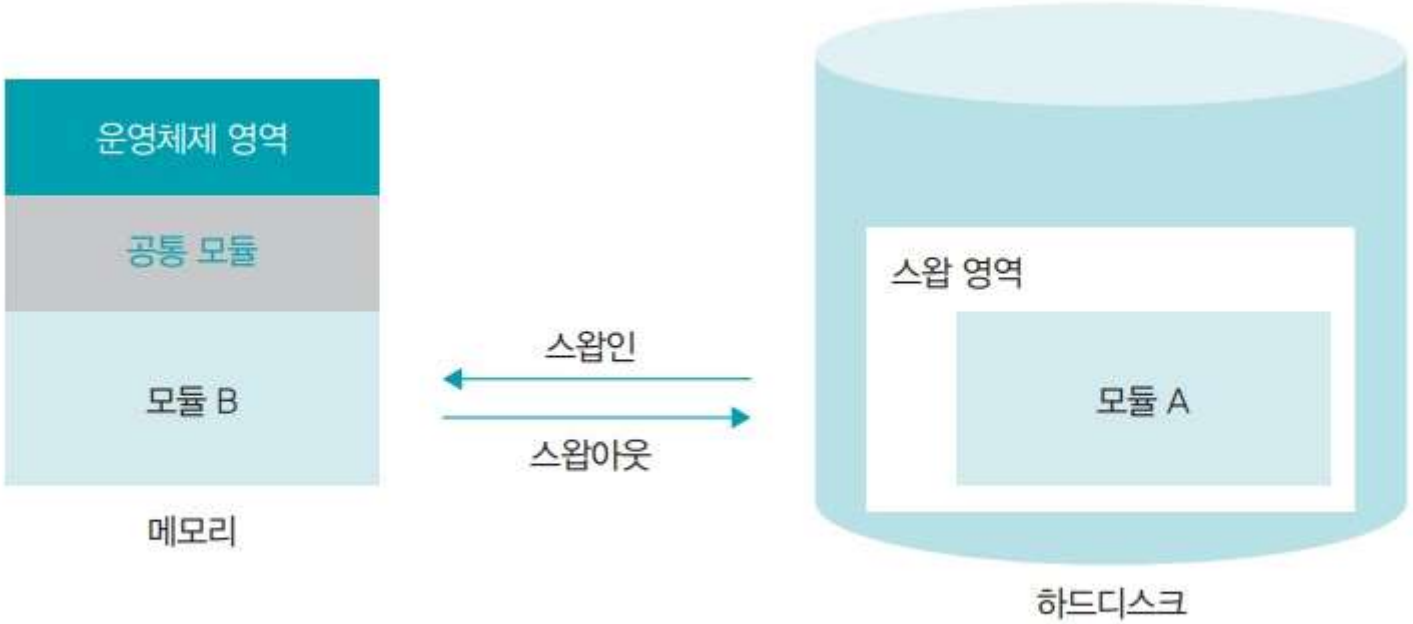


그림 7-17 스왑인과 스왑아웃

1 메모리 분할 방식

메모리에 여러 개의 프로세스를 배치하는 방법

- 고정 분할 방식 **fixed-size partitioning**
  - 프로세스의 크기와 상관없이 메모리를 같은 크기로 나누는 것
- 가변 분할 방식 **variable-size partitioning**
  - 프로세스의 크기에 따라 메모리를 나누는 것

식당 의자 비유

- 고정 분할 방식 : 손님의 신체 크기와 상관없이 같은 크기의 의자를 준비하는 것
- 가변 분할 방식 : 손님의 신체 크기에 맞게 의자를 준비하는 것

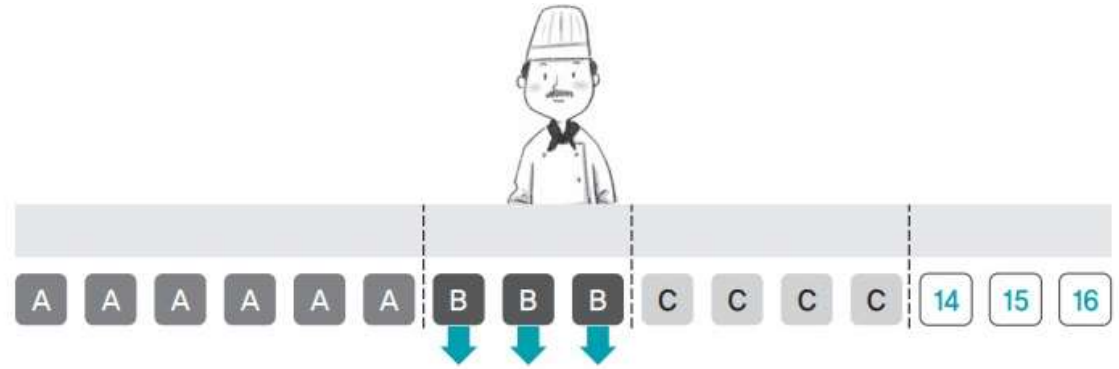
(a) 가변 분할 방식　　(b) 고정 분할 방식

그림 7-22 식당 의자에 비유한 가변 분할 방식과 고정 분할 방식

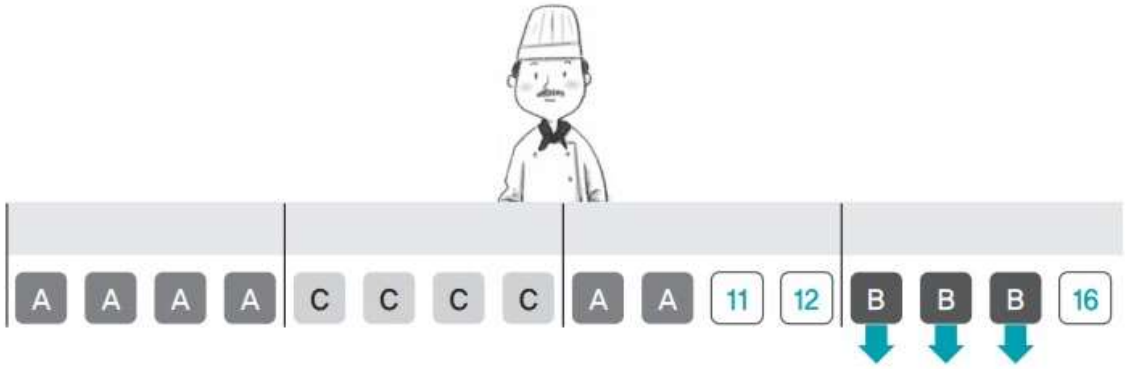
1 메모리 분할 방식

손님 자리 배치 비유

- 가변 분할 방식 : 어디에 앉든 제약이 없기 때문에 손님들이 알아서 편한 자리에 앉음
- 고정 분할 방식 : 의자를 4개씩 파티션으로 나누고 손님들의 자리를 파티션 단위로 배정



(a) 가변 분할 방식(손님 B가 식사를 마침)



(b) 고정 분할 방식(손님 B가 식사를 마침)

그림 7-23 **손님 자리 배치에 비유한 가변 분할 방식과 고정 분할 방식**

1 메모리 분할 방식

메모리 분할 방식의 구현

- 가변 분할 방식
  - 프로세스의 크기에 맞게 메모리가 분할
  - 메모리의 영역이 각각 다름
  - 연속 메모리 할당
- 고정 분할 방식
  - 프로세스의 크기에 상관없이 메모리가 같은 크기로 나뉨
  - 큰 프로세스가 메모리에 올라오면 여러 조각으로 나누어 배치
  - 비연속 메모리 할당

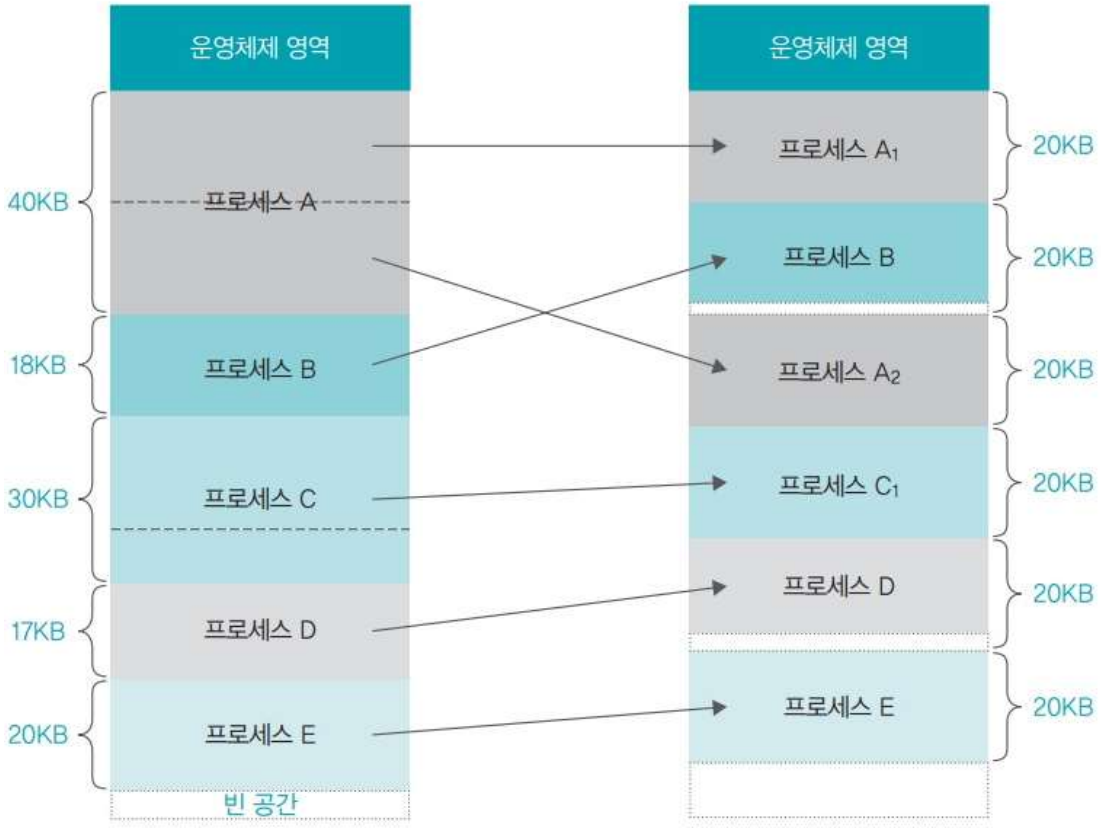


그림 7-24 메모리 분할 방식의 구현

1 메모리 분할 방식

가변 분할 방식의 장단점

- 장점 : 프로세스를 한 덩어리로 처리하여 하나의 프로세스를 연속된 공간에 배치
- 단점 : 비어 있는 공간을 하나로 합쳐야 하며, 이 과정에서 다른 프로세스의 자리도 옮겨야 하므로 메모리 관리가 복잡함

고정 분할 방식의 장단점

- 장점 : 메모리를 일정한 크기로 나누어 관리하기 때문에 메모리 관리가 수월(가변 분할 방식의 메모리 통합 같은 부가적인 작업을 할 필요가 없음)
- 단점 : 쓸모없는 공간으로 인해 메모리 낭비가 발생할 수 있음

단편화 fragmentation

- 메모리가 여러 작은 조각/부분들로 나뉘어지는 현상

고정 분할 방식의 내부 단편화

- 스케줄링과 분할 크기에 따른 내부 단편화의 변화

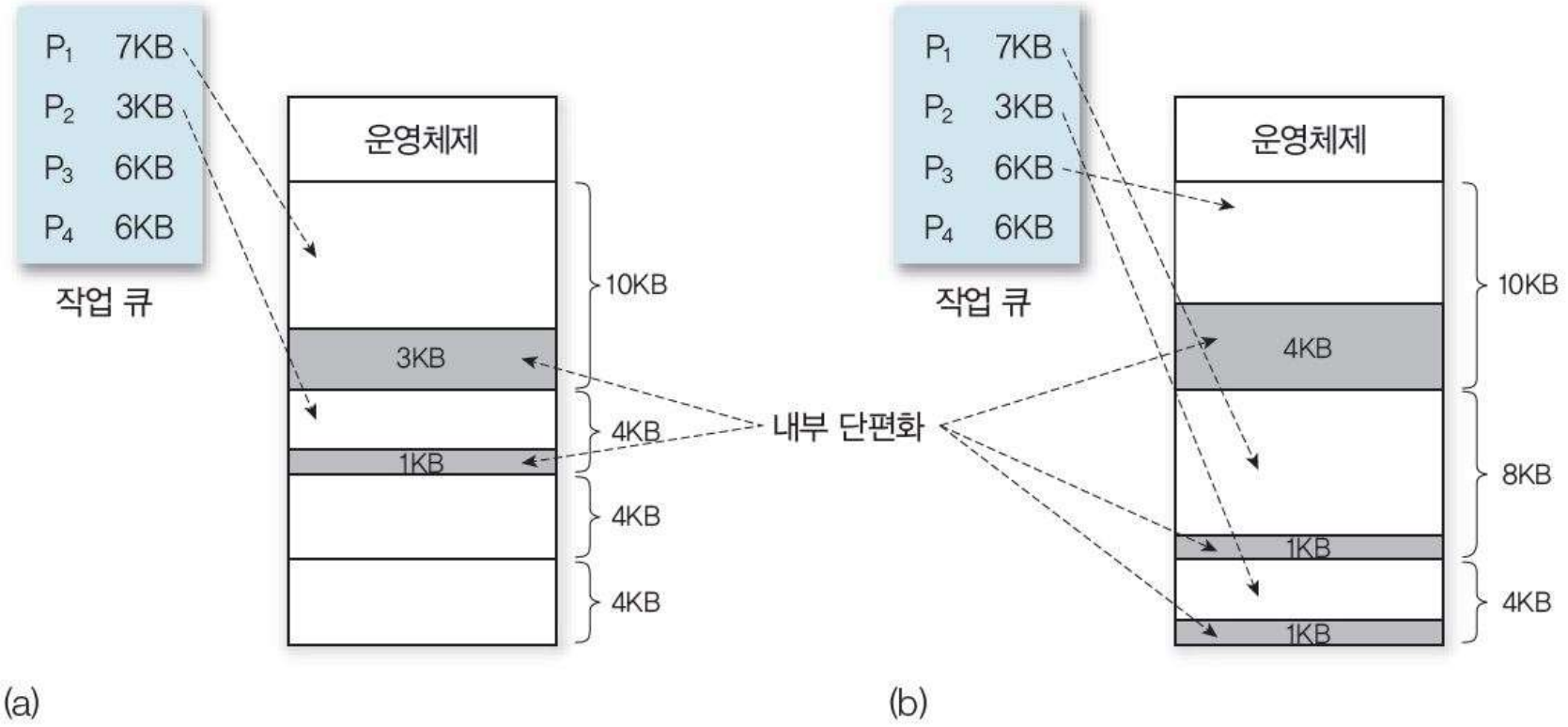


그림 7-14 스케줄링과 분할 크기에 따른 내부 단편화의 변화

2 가변 분할 방식의 메모리 관리

가변 분할 방식과 외부 단편화

- 프로세스 A, B, C, D, E를 순서대로 배치했을 때 프로세스 B와 D가 종료되면 18KB와 17KB의 빈 공간이 생김
- 이후 18KB보다 큰 프로세스가 들어오면 적당한 공간이 없어 메모리를 배정하지 못하는데, 가변 분할 방식에서 발생하는 이러한 작은 빈 공간을 외부 단편화라고 함

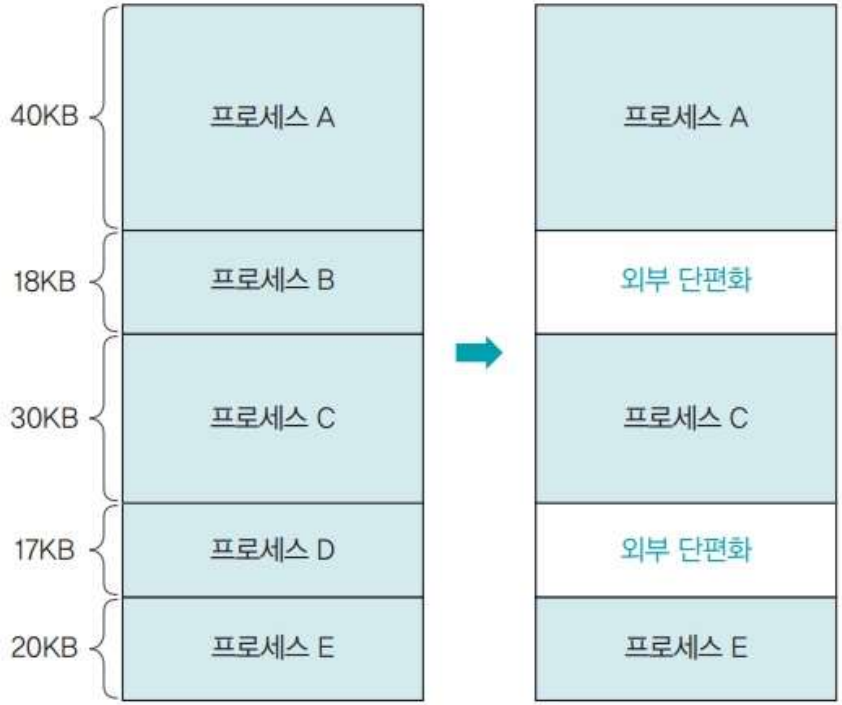


그림 7-28 가변 분할 방식과 외부 단편화

2 가변 분할 방식의 메모리 관리

외부 단편화 external fragmentation 해결

- 메모리 배치 방식 : 작은 조각이 발생하지 않도록 프로세스를 배치하는 것
- 조각 모음 : 조각이 발생했을 때 작은 조각들을 모아서 하나의 큰 덩어리로 만드는 작업

메모리 배치 방식

- 최초 배치first fit
  - 프로세스를 메모리의 빈 공간에 배치할 때 메모리에서 적재 가능한 공간을 순서대로 찾다가 첫 번째로 발견한 공간에 프로세스를 배치하는 방법
- 최적 배치best fit
  - 메모리의 빈 공간을 모두 확인한 후 적당한 크기 가운데 가장 작은 공간에 프로세스를 배치하는 방법
- 최악 배치worst fit
  - 빈 공간을 모두 확인한 후 가장 큰 공간에 프로세스를 배치하는 방법

2 가변 분할 방식의 메모리 관리

메모리 배치 방식

그림 7-29 **메모리 배치 방식**

2 가변 분할 방식의 메모리 관리

메모리 배치 방식 비교

그림 7-30 **메모리 배치 방식의 비교**

2 가변 분할 방식의 메모리 관리

메모리 배치 방식 비교

- 최초 배치
  - 빈 공간을 찾아다닐 필요 없음
- 최적 배치
  - 빈 공간을 모두 확인하는 부가적인 작업이 있지만 딱 맞는 공간을 찾을 경우 단편화가 일어나지 않음
  - 딱 맞는 공간이 없을 때는 아주 작은 조각을 만들어내는 단점이 있음
- 최악 배치
  - 프로세스를 배치하고 남은 공간이 크기 때문에 쓸모가 있음
  - 빈 공간의 크기가 클 때는 효과적이지만 빈 공간의 크기가 점점 줄어들면 최적 배치처럼 작은 조각을 만들어냄

2 가변 분할 방식의 메모리 관리

조각 모음defragmentation / 메모리 압축

- 이미 배치된 프로세스를 옆으로 옮겨 빈 공간들을 하나의 큰 덩어리로 만드는 작업
- 조각 모음 순서

❶ 조각 모음을 하기 위해 이동할 프로세스의 동작을 멈춤
❷ 프로세스를 적당한 위치로 이동(프로세스가 원래의 위치에서 이동하기 때문에 프로세스의 상대 주소값을 바꿈)
❸ 작업을 다 마친 후 프로세스 다시 시작

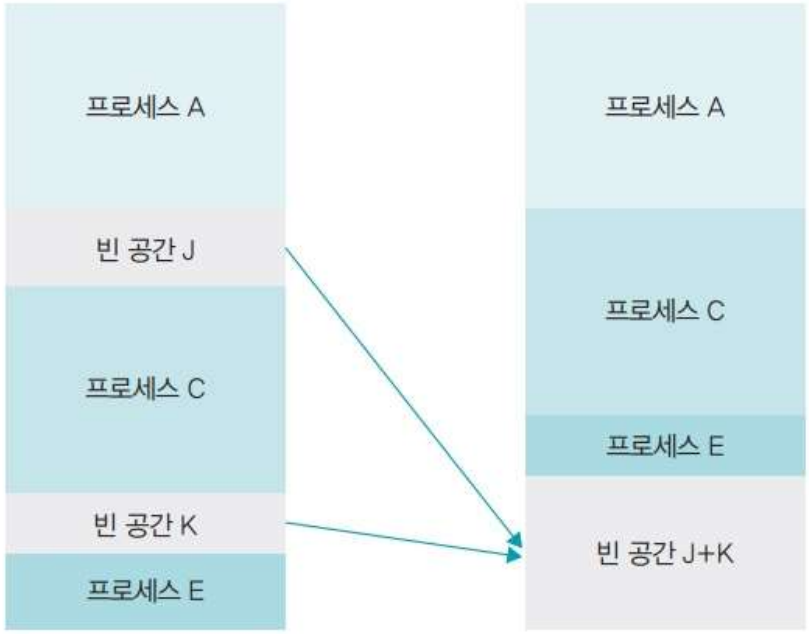


그림 7-31 조각 모음

3 고정 분할 방식의 메모리 관리

고정 분할 방식의 프로세스 배치

- 분할된 크기는 20KB이므로 40KB인 프로세스 A는 프로세스 A1과 A2로 나뉘어 메모리에 할당
- 30KB인 프로세스 C는 프로세스 C1과 C2로 나뉘는데, 메모리에 남은 공간이 없으므로 프로세스 C2는 스왑 영역으로 옮겨짐

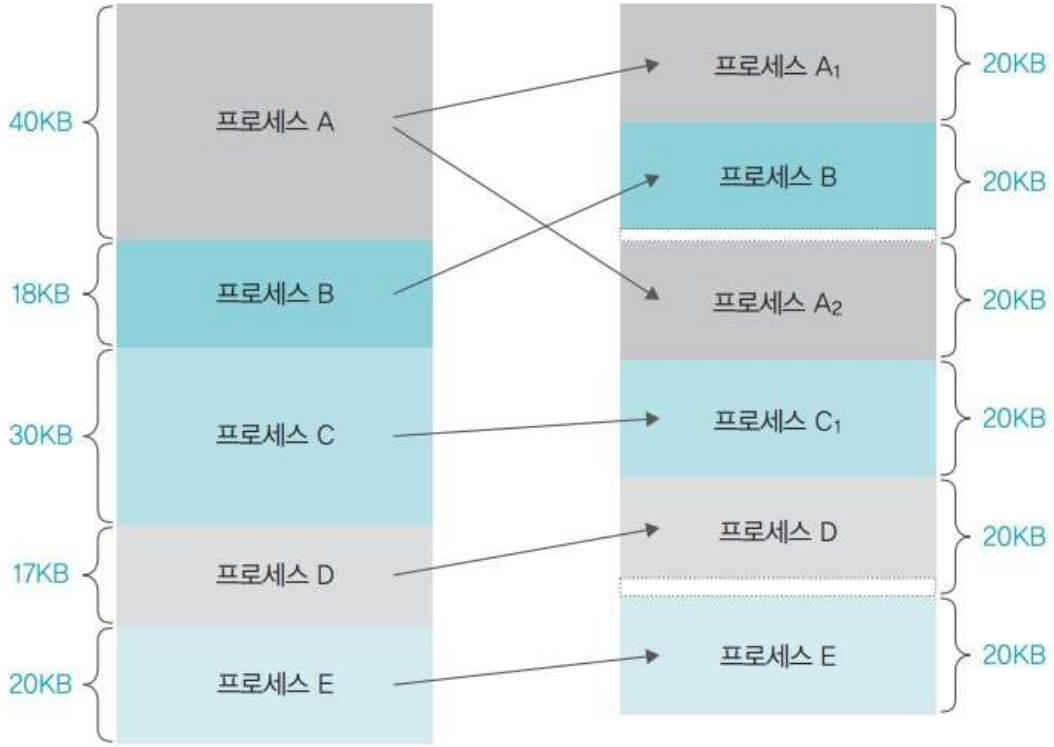


그림 7-34 고정 분할 방식의 프로세스 배치

3 고정 분할 방식의 메모리 관리

고정 분할 방식과 내부 단편화 internal fragmentation

- 각 메모리 조각에 프로세스를 배치하고 공간이 남는 현상
- 고정 분할 방식은 내부 단편화를 줄이기 위해 신중하게 메모리의 크기를 결정해서 나눠야 하지만 사용하는 프로세스의 크기가 제각각이기 때문에 메모리를 얼마로 나누느냐에 관한 정답은 없음

가변 분할 방식과 고정 분할 방식의 비교

표 7-4 가변 분할 방식과 고정 분할 방식의 비교

구분	가변 분할 방식	고정 분할 방식
메모리 단위	세그먼테이션	페이징
특징	연속 메모리 할당	비연속 메모리 할당
장점	프로세스를 한 덩어리로 관리 가능	메모리 관리가 편리
단점	빈 공간의 관리가 어려움	프로세스가 분할되어 처리됨
단편화	외부 단편화	내부 단편화

4 버디 시스템

버디 시스템의 작동 방식

❶ 프로세스의 크기에 맞게 메모리를 ½로 자르고 프로세스를 메모리에 배치
❷ 나뉜 메모리의 각 구역에는 프로세스가 1개만 들어감
❸ 프로세스가 종료되면 주변의 빈 조각과 합쳐서 하나의 큰 덩어리를 만듦

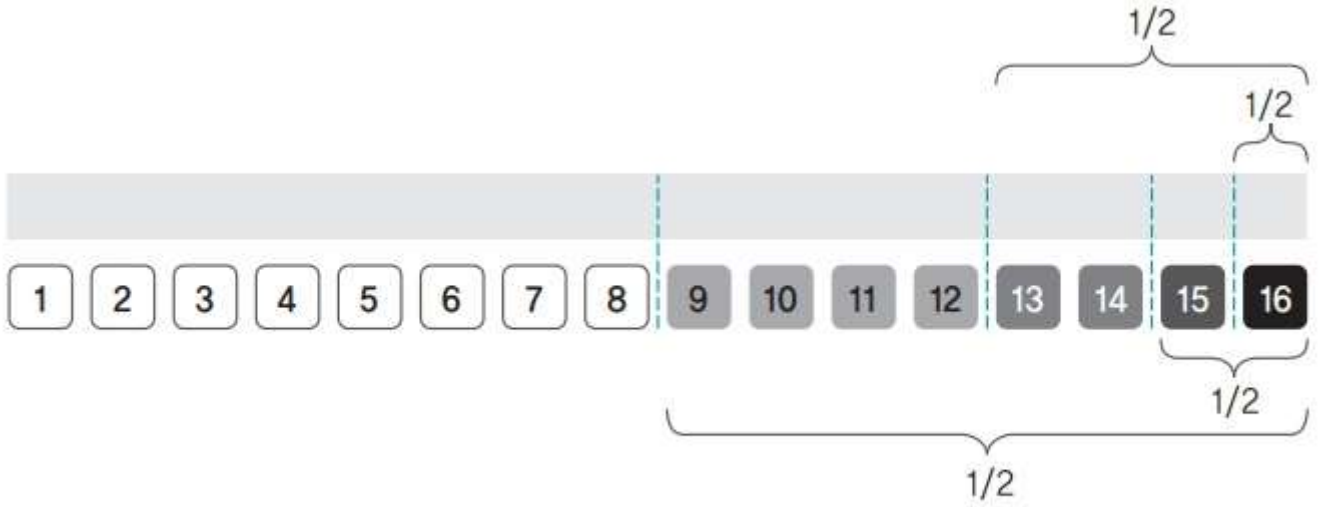


그림 7-35 버디 시스템의 메모리 분할 예

4 버디 시스템

버디 시스템의 자리 배정(일식집 예)

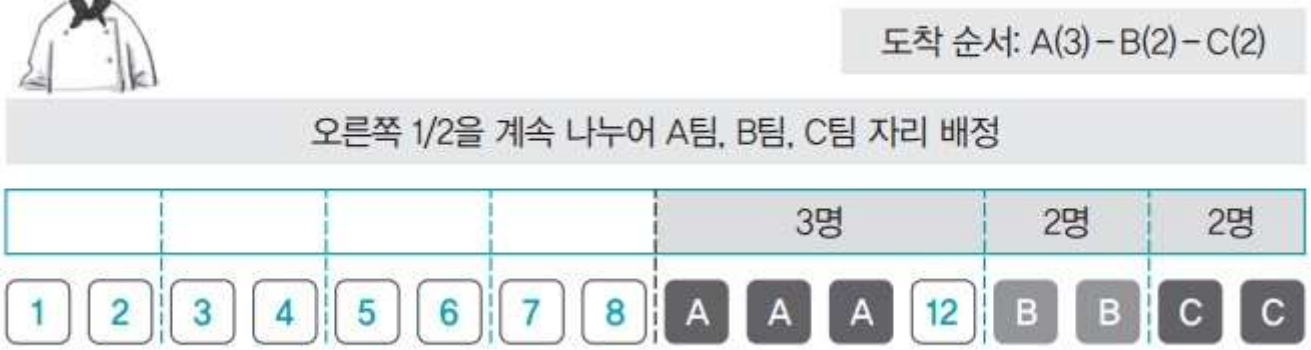


그림 7-36 버디 시스템의 자리 배정 예 1

4 버디 시스템

버디 시스템 특징

- 가변 분할 방식처럼 메모리가 프로세스 크기대로 나뉨
- 고정 분할 방식처럼 하나의 구역에 다른 프로세스가 들어갈 수 없고, 메모리의 한 구역 내부에 조각이 생겨 내부 단편화 발생
- 비슷한 크기의 조각이 서로 모여 작은 조각을 통합하여 큰 조각을 만들기 쉬움

단편화(fragmentation)

- **내부 단편화(Internal fragmentation)**
  - 분할된 메모리 크기에 할당된 작업이 적재되고 남겨지는 공간
- **외부 단편화(External fragmentation)**
  - 메모리 할당을 요청한 작업보다 메모리에 남은 공간이 더 많더라도 연속적인 공간이 아니여서 해당 작업을 메모리에 적재하지 못하는 경우 남은 메모리 공간

Homework

7장 연습문제

- P.374 연습문제(단답형)
- P.376 심화문제(서술형)
- 제출기한 : 12월 1일(일) 23시
- 제출방법 : hwp/word로 작성한 후 cyber 과제함
- 파일이름 : **학번(이름)-7장**.hwp/doc/pdf

Thank You

1 컴파일 과정

컴파일 과정

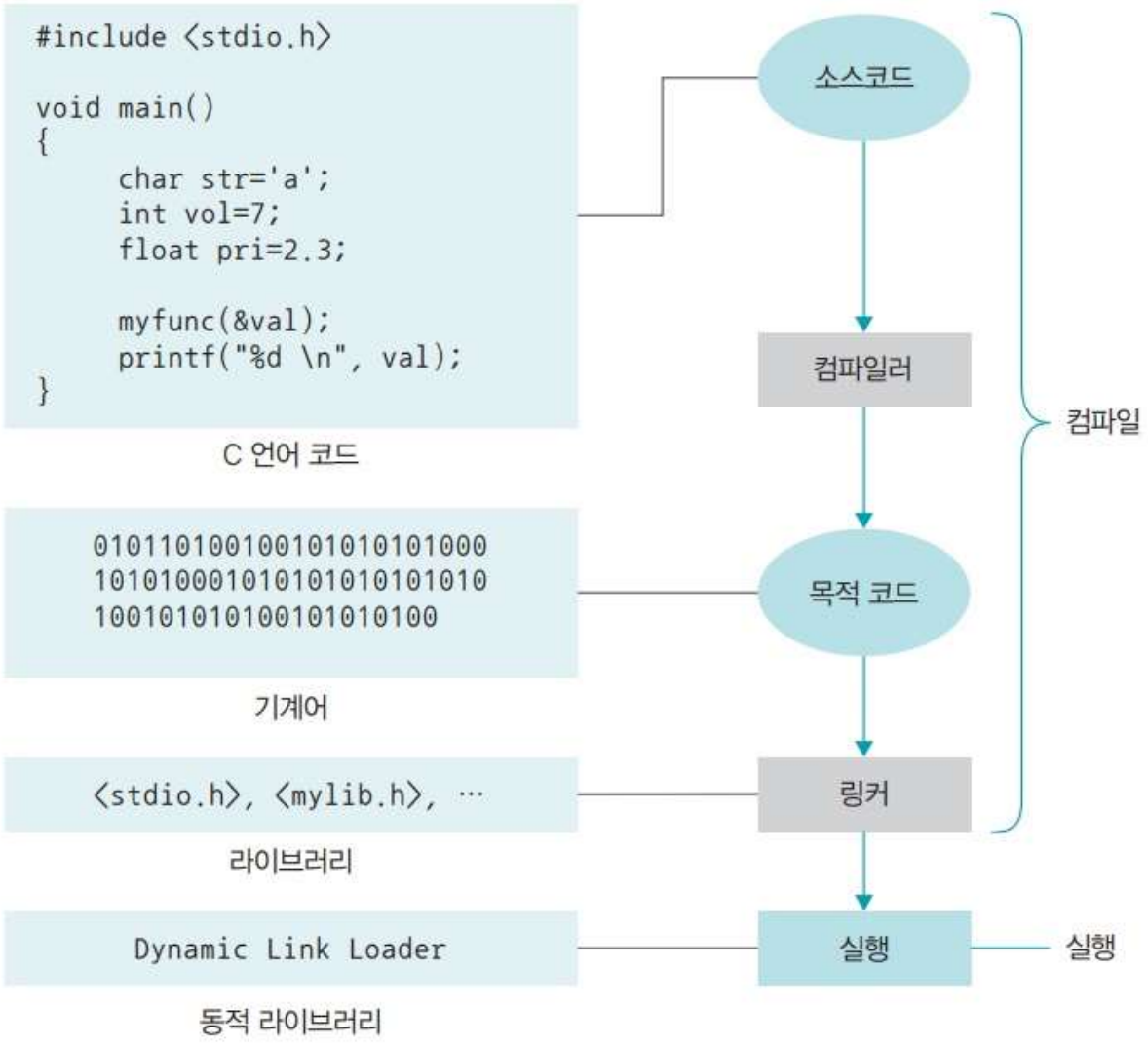


그림 7-39 **컴파일 과정**

2 변수와 메모리 할당

변수와 메모리

소스코드 7-1 코드 선언부

```
01  char str='a';
02  int vol=7;
03  float pri=2.3;
```

- 01행 : 변수 str을 문자형으로 선언하고 그곳에 a를 넣으라는 뜻
- 02행 : 변수 vol을 정수형으로 선언하고 그곳에 7을 넣으라는 뜻
- 03행 : 변수 pri를 실수형으로 선언하고 그 곳에 2.3을 넣으라는 뜻

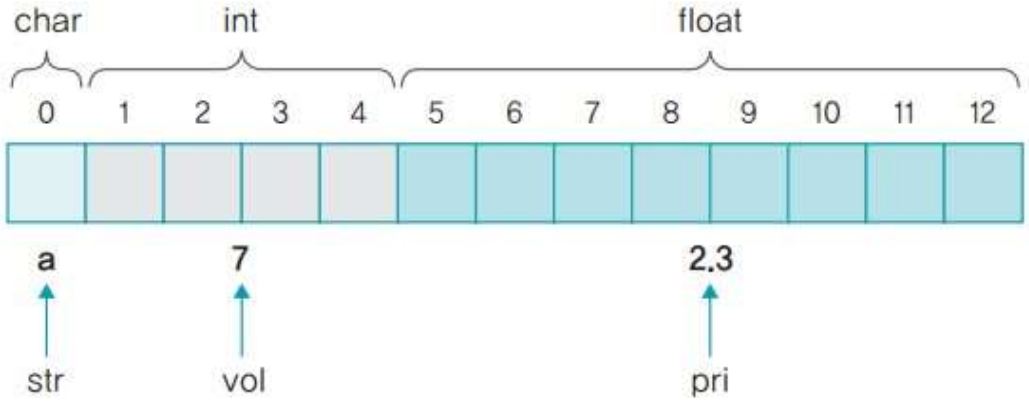


그림 7-40 변수와 메모리

2 변수와 메모리 할당

컴파일러와 변수

- 컴파일러는 모든 변수에 대해 메모리를 확보하고 오류를 찾기 위해 심벌 테이블 유지

표 7-5 심벌 테이블의 예

심벌	종류	범위	주소
str	char	main()	0
vol	int	main()	1
pri	float	main()	5

- 컴파일러는 변수를 사용할 때마다 사용 범위를 넘는지 점검
- 컴파일러는 모든 변수를 메모리 주소로 바꾸어 기계어로 된 실행 파일을 만듦
- 컴파일러에 의해 만들어진 변수의 주소는 상대주소임