컴퓨터학개론

5주차

최종석(jschoi@ssu.ac.kr)

02
운영체제의 개요

02. 운영체제의 개요

I. 운영체제의 개념

- **운영체제(OS, Operating System) :** 하드웨어 장치를 관리하고 사용자가 컴퓨터를 편하게 사용할 수 있는 환경을 제공하며, 응용 소프트웨어의 효율적인 실행을 지원하는 시스템 소프트웨어.

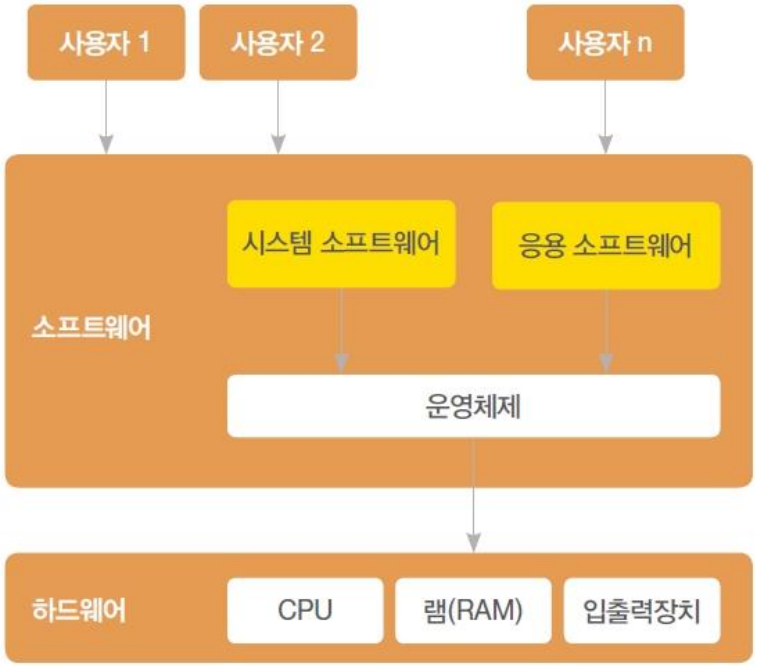


그림 4-4 운영체제의 위치

02. 운영체제의 개요

I. 운영체제의 개념

하나 더 알기 **컴퓨터를 켜면 내부에서는 어떤 일이 일어날까?**

- 전원을 눌러 컴퓨터가 실행되면 롬(ROM)에 저장된 프로그램인 부트스트랩 로더가 하드디스크에 있는 운영체제 실행을 도움.
- **컴퓨터의 부팅 과정**

① 컴퓨터 전원을 켜면 외부의 전압이 내부에서 사용할 수 있는 전압으로 변환됨. 이 전기는 CPU로 전달됨. CPU 레지스터인 프로그램 카운터 레지스터는 메인보드 내 롬 바이오스(ROM BIOS)의 부트 프로그램 주소 값으로 초기화.

② 부트 프로그램은 CPU의 이상 유무를 테스트한 후, 테스트 결과가 롬 바이오스에 저장된 값과 일치하면 포스트 작업 수행.

02. 운영체제의 개요

I. 운영체제의 개념

하나 더 알기 **컴퓨터를 켜면 내부에서는 어떤 일이 일어날까?**

- **컴퓨터의 부팅 과정**

③ 부트 프로그램은 운영체제를 로드하기 위해 하드디스크의 첫 번째 섹터에 있는 마스터 부트 레코드(MBR)를 바탕으로 운영체제의 위치를 찾아 메인 메모리에 적재. 이때부터 운영체제에서 규정한 부팅 과정이 시작됨.

④ 운영체제는 이니트(Init)와 같은 첫 번째 프로세스를 실행하고, 사용자로부터 요청이 발생하기를 기다림.

✓ **TIP. 포스트(POST) :** 시스템 버스, 실시간 시계(RTC), 비디오 구성 요소들, 램(RAM), 키보드, 연결된 드라이브에 신호를 보내 컴퓨터가 정상적으로 동작하는지 점검하는 과정.

02. 운영체제의 개요

II. 운영체제의 기능

▪ 자원 관리

- 컴퓨터의 멀티태스킹 환경이 보편화되면서 운영체제의 CPU 스케줄링 및 메모리 관리 기능도 점점 복잡해지고 있음.
- 개인용 컴퓨터에서 응용 프로그램을 실행해야 할 때 어떤 프로그램을 실행할지 결정하는 것이 바로 운영체제의 역할임.

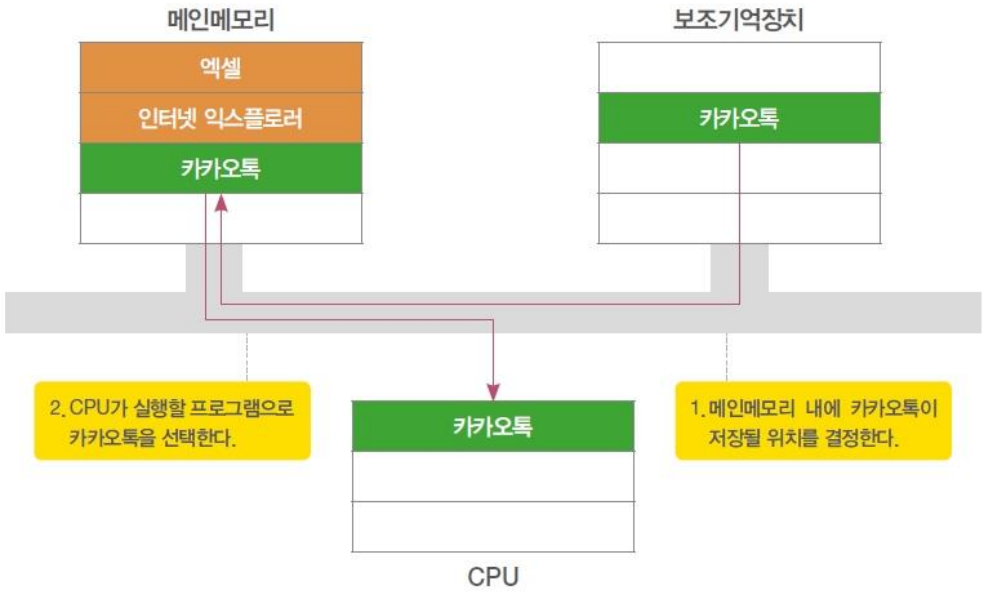


그림 4-5 운영체제의 자원 관리

02. 운영체제의 개요

II. 운영체제의 기능

▪ 자원 보호

- 컴퓨터의 내부 자원은 악의적인 사용자나 미숙한 사용자로부터 보호를 받아야 함.
- 운영체제는 사용자가 사용하는 프로그램에 오류가 발생하거나 자원이 잘못 사용되는 것을 감시해 컴퓨터 자원을 지키는 역할도 수행.

▪ 사용자 인터페이스 제공

- **사용자 인터페이스(User Interface) :** 사용자와 컴퓨터 간에 상호작용이 이루어지는 환경으로, 최근에는 사용자가 컴퓨터를 더 쉽고 효율적으로 사용할 수 있도록 멀티터치 환경이나 3D 인터페이스 환경 등을 제공하고 있음.

02. 운영체제의 개요

III. 운영체제의 종류

▪ 컴퓨터 운영체제

- **유닉스(UNIX)**
  - 1969년, 벨(Bell) 연구소의 프로그래머였던 켄 톰슨이 개발.
  - 현재는 기업, 대학, 연구기관 등에서 널리 사용되고 있음.
  - 다양한 종류의 컴퓨터에서 작동되며, 인터넷 서버 컴퓨터의 핵심으로 자리함.

- **리눅스(LINUX)**
  - 1991년, 리누스 토발즈가 개발한 유닉스 호환 운영체제.
  - 소스코드를 공개하여 사용자가 자유롭게 이용할 수 있음.
  - 다중 사용자·다중 작업·가상 터미널 환경 지원, 네트워크 운영체제 활용 가능.

02. 운영체제의 개요

III. 운영체제의 종류

▪ 컴퓨터 운영체제

- **윈도우(Windows)**
  - 마이크로소프트사가 개발한 운영체제로, 개인용 컴퓨터에서 가장 많은 사용자층을 확보하고 있음.
  - 도스 운영체제(DOS)에서 꾸준히 업그레이드되어 시스템이 상당히 안정화됨.

- **맥 OS(Mac OS)**
  - 애플사의 개인용 컴퓨터인 매킨토시에 사용할 목적으로 개발된 그래픽 사용자 인터페이스 운영체제.
  - 강력한 기능의 소프트웨어, 사용자 친화적인 디자인, 쉬운 사용법, 스마트 기기와의 연계를 통한 일관된 통합 환경 구축

III. 운영체제의 종류

▪ 모바일 운영체제

- **안드로이드(Android) :** 미들웨어, 사용자 인터페이스, 표준 응용 프로그램을 포함한 소프트웨어 스택이자 모바일 운영체제로, 리눅스 커널 위에서 동작하며, 다양한 안드로이드 시스템 구성요소에서 사용되는 Java, C/C++ 라이브러리들을 포함하고 있음.
- **iOS :** 애플사의 아이폰과 아이패드에 내장된 모바일 운영체제로, 맥 OS를 기반으로 함

(a) 리눅스 (b) 윈도우 (c) 맥 OS (d) 안드로이드 (e) iOS

그림 4-6 운영체제 종류

IV. 운영체제의 구성

▪ 커널(Kernel)

- **커널 :** 운영체제의 가장 핵심적인 부분으로, 일반 사용자가 관여하지 못하는 시스템 레벨 수준의 제어 작업을 수행함. 커널은 메모리 관리, 프로세스 관리, 입출력장치 관리 등의 일을 처리함.

▪ 사용자 인터페이스(User Interface)

- **사용자 인터페이스 :** 컴퓨터 사용자가 직접 프로그램을 제어하고 사용할 수 있는 환경으로, 운영체제가 커널에 명령을 전달하고 실행 결과를 사용자와 응용 프로그램에 돌려주는 방식임. GUI와 CLI 방식으로 나눌 수 있음.

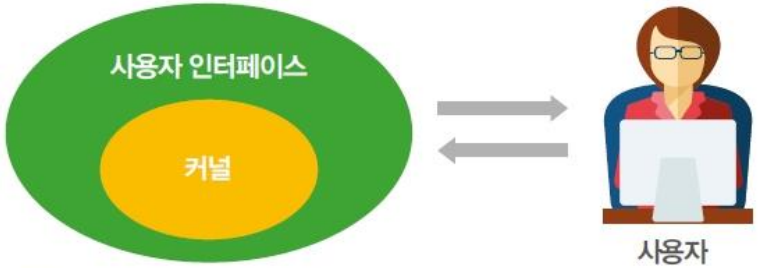


그림 4-7 운영체제의 커널과 사용자 인터페이스

V. 운영체제의 관리

- 초기 컴퓨터는 한번에 하나의 프로그램만 실행되는 단일 프로그래밍 시스템이었음.
- 오늘날은 다중 프로그래밍 시스템이 주로 사용되면서 운영체제의 프로세스 및 메모리 관리 기능이 중요해짐.
- 운영체제는 이런 상황에서 여러 개의 프로세스가 서로 충돌하지 않도록 메모리상의 프로세스들을 잘 관리해야 하며, 메모리 내 프로세스의 위치를 추적하기 위해 메모리 관리도 수행해야 함.

03
운영체제의 프로세스 관리

03. 운영체제의 프로세스 관리

I. 프로세스의 개념

- **프로세스(Process) :** 실행 명령을 받아 메인메모리로 올라간 프로그램으로, 컴퓨터상에서 실행 중인 프로그램을 뜻함.
- 프로세스는 운영체제에서 운영되는 시스템 작업의 기본 단위.

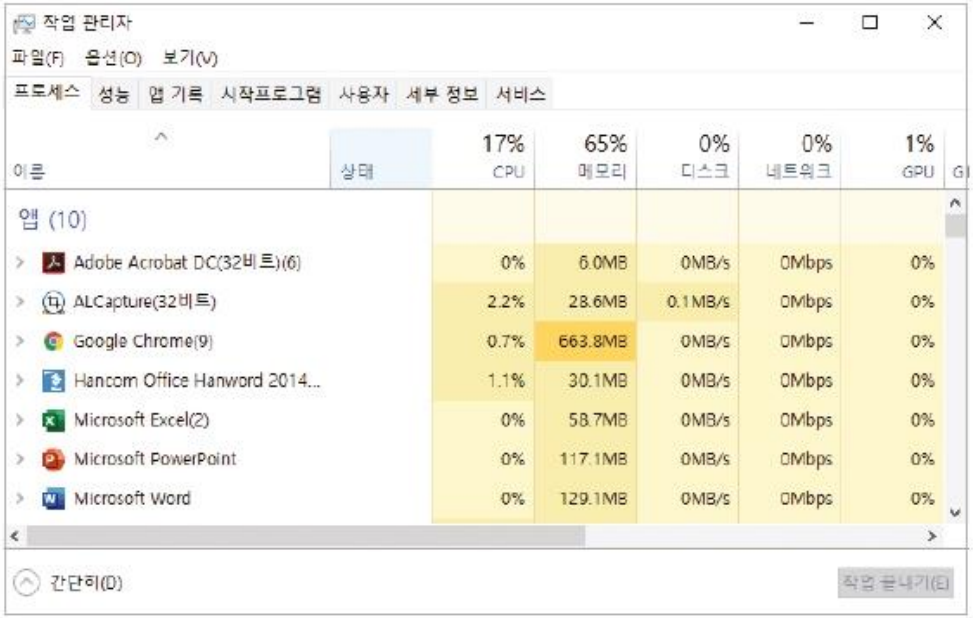


그림 4-8 윈도우 운영체제에서의 실행 프로세스 예

03. 운영체제의 프로세스 관리

I. 프로세스의 개념

- **여러 개의 프로세스가 동시에 실행되는 과정**

① 준비(Ready) : 생성된 프로세스가 프로세서인 CPU 사용 시간을 할당받기 위해 기다리고 있는 상태.

② 실행(Running) : CPU에 의해 프로세스가 실행되고 있는 상태.

- 종료 : 프로세스의 모든 작업이 끝나면 종료됨.
- 중단/방해 : 프로세스의 CPU 할당 시간이 완료되거나 방해가 발생하면 준비 상태로 넘어감.
- 입출력 또는 이벤트 기다림 : 실행 중인 프로세스가 입출력 명령을 마치면 CPU 사용을 반납하고 입출력 종료 신호가 올 때까지 대기 상태로 넘어감.

03. 운영체제의 프로세스 관리

I. 프로세스의 개념

- **여러 개의 프로세스가 동시에 실행되는 과정**

③ 대기(Blocked) : 입출력 등과 같이 임의의 자원을 요청한 후 할당받을 때까지 기다리는 상태.

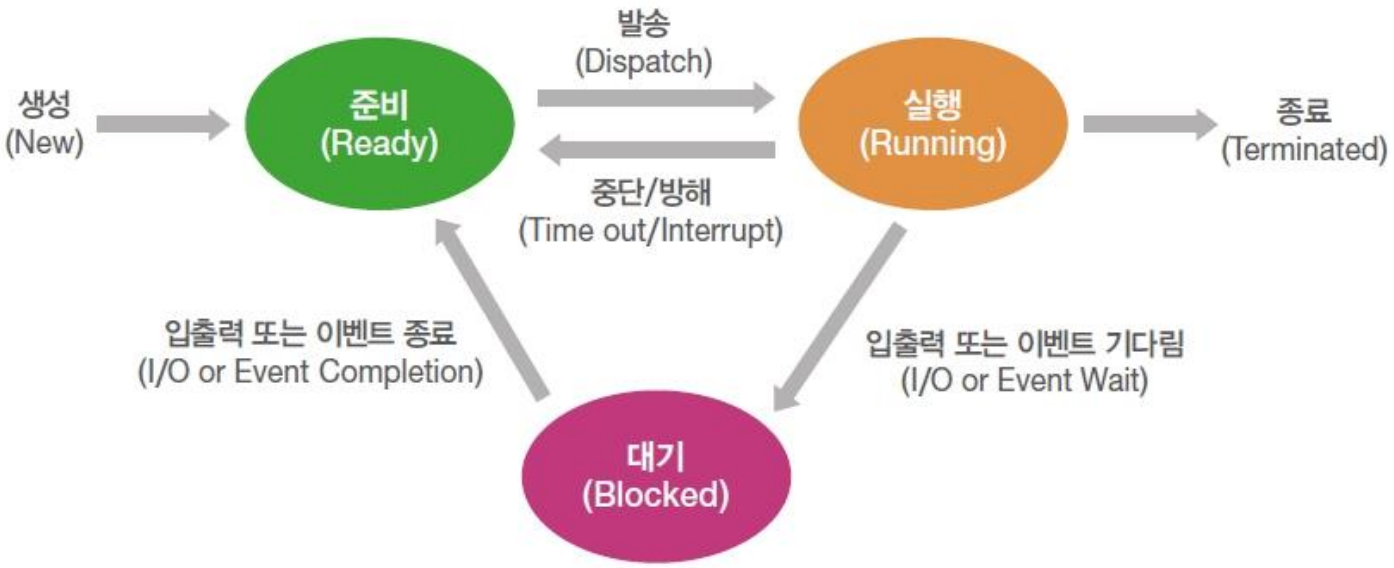


그림 4-9 프로세스 상태도 : 다수의 프로세스 관리하기

03. 운영체제의 프로세스 관리

II. 프로세스 제어 블록

- **프로세스 제어 블록(PCB) :** 운영체제가 관리하는 자료구조로, 프로세스 관리에 필요한 거의 모든 정보가 저장되어 있는 장소임.
- 기본적으로 프로세스 ID, 각 프로세스에서 사용하는 레지스터 메모리 값의 백업 값 등이 저장되어 있는데, 모든 프로세스는 별도의 PCB를 가지며 프로세스가 생성될 때 만들어졌다가 프로세스가 실행을 마치면 삭제됨.

그림 4-10 프로세스 제어 블록(PCB)의 구성

03. 운영체제의 프로세스 관리

II. 프로세스 제어 블록

- **문맥 전환(Context Switching) :** 다중 프로그래밍 환경에서 실행 상태인 프로세스가 사용하는 레지스터 값 등을 해당 프로세스의 PCB에 저장하고, 새롭게 실행 상태로 넘어온 프로세스의 레지스터 값 등은 CPU에 적재하는 과정.

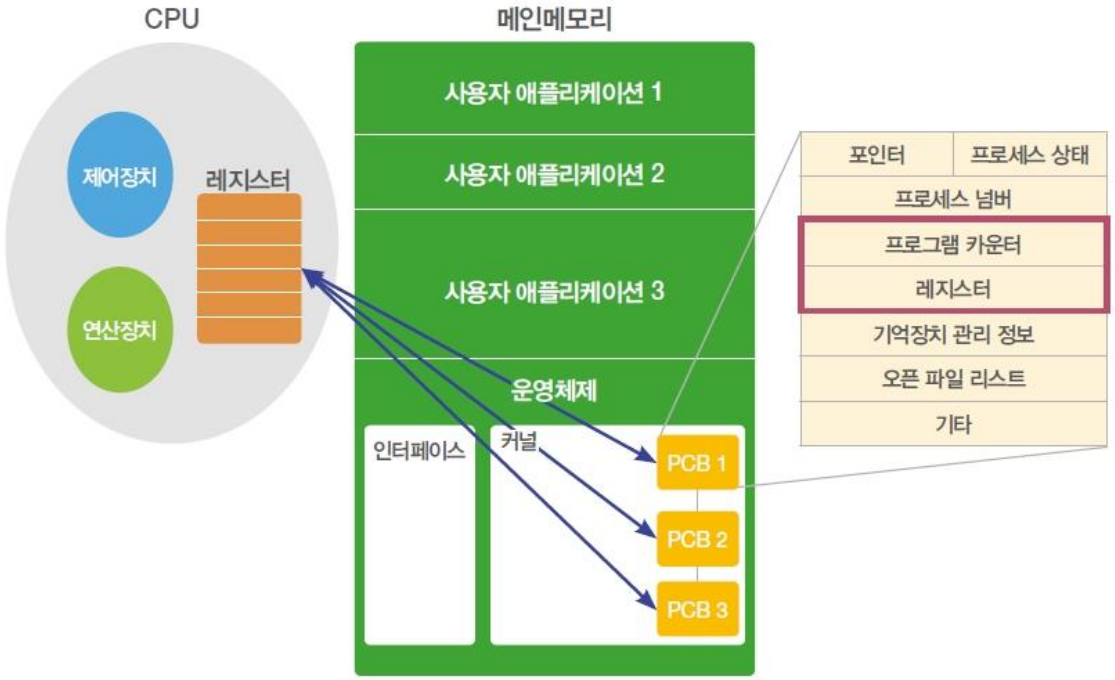


그림 4-11 PCB의 문맥 전환의 개념

03. 운영체제의 프로세스 관리

III. 프로세스 스케줄링

- **프로세스 스케줄링(Process Scheduling) :** 다중 프로그래밍 환경에서 현재 실행 중인 프로세스 다음에 어떤 프로세스를 실행하면 좋을지 결정하는 과정.
- **프로세스 스케줄링 기법 3가지**
  - 선도착 선처리 기법(FCFS) : 먼저 생성된, 즉 먼저 준비 상태로 들어온 프로세스를 먼저 실행시킴.
  - 최단작업 우선 기법(SJF) : 준비 상태에 있는 프로세스 중 CPU 사용 시간이 가장 짧은 것부터 실행시키는 방법
  - 순환 순서 기법(Round Robin): 여러 프로세스가 돌아가면서 CPU를 조금씩 차지하는 방법.

03. 운영체제의 프로세스 관리

IV. 병행 프로세스

- **병행 프로세스(Concurrent Processes) :** 2개 이상의 프로세스가 동시에 실행되는 것으로, 다른 프로세스와 협력하면서 실행되므로 서로 영향을 주고받기 때문에 수행 과정상 예측할 수 없는 상황이 발생하기도 함.
- **병행 프로세스의 오류를 막기 위해 사용하는 개념**
  - 동기화(Synchronization) : 하나의 자원(데이터)에 2개 이상의 프로세스가 동시에 접근했을 때 처리 순서를 잘 정해 일관성을 유지해야 함.
  - 임계 영역(Critical Section) : 병행 프로세스의 코드 영역 중 자원(데이터)을 읽고 수정하는 등의 작업이 이루어지는 부분.
  - 상호 배제(Mutual Exclusion) : 하나의 프로세스가 공유 자원을 사용하는 동안 다른 프로세스가 자원을 사용하지 못하게 막는 것.

IV. 병행 프로세스

하나 더 알기 **임계 영역**

- 고급 프로그래밍 언어로 작성된 프로그램은 기계어로 번역되어 실행되는데, 만약, 프로세스 1과 프로세스 2로 이루어진 병행 프로세스가 동일한 변수인 number를 사용한다면?
- 프로그래밍 언어 측면에서 이 2개의 프로세스는 병행 수행되더라도 별 문제가 없어 보이는데, 이는 어떤 프로세스가 먼저 실행되더라도 변수 값은 최종적으로 2가 증가할 것으로 예상되기 때문.

```
...
number=number + 1
...
```

[프로세스 1]

```
...
number=number + 1
...
```

[프로세스 2]

- 그런데 어셈블리어 또는 기계어 측면에서 보면 상황은 달라짐.

03. 운영체제의 프로세스 관리

IV. 병행 프로세스

하나 더 알기 **임계 영역**

- 컴퓨터에서 메인메모리의 특정 위치 값을 변경하려면 메인메모리에 저장되어 있는 변수 값을 CPU로 로드LDA)한 후에, CPU 내부에서 1을 증가시키고(ADDA), 그 결과 값을 다시 메인메모리에 저장하는(STA) 과정을 거쳐야 하기 때문.
- 고급 프로그래밍 언어에서는 하나의 명령어처럼 보여도 기계어 수준에서는 여러 개의 명령어로 분할될 수 있음.

```
...
(P1-1) LDA number, d
(P1-2) ADDA 1, i
(P1-3) STA number, d
...
```

[프로세스 1]

```
...
(P2-1) LDA number, d
(P2-2) ADDA 1, i
(P2-3) STA number, d
...
```

[프로세스 2]

- 또한 어셈블리어 프로그램 2개가 모두 수행을 마치면 number 변수는 2가 증가되어야 하는데, 실제로는 1밖에 증가되지 않는 상황이 발생할 수 있음.

IV. 병행 프로세스

하나 더 알기 **임계 영역**

- 2개의 프로세스 명령어가 엇갈려서 하나씩 수행되면 number 변수가 최종적으로는 1만 증가함. 공유 변수인 number 변수를 특정 프로세스가 사용하고 있을 때 다른 프로세스가 끼어들어 사용하기 때문임.

  • 수행 순서 예 : P1-1 / P1-2 / P2-1 / P2-2 / P2-3 / P1-3

- number = number + 1과 같이, 공유 변수를 사용하는 코드 영역은 프로세스 1과 프로세스 2에 의한 상호 배제 방식으로 수행되어야 함.
- **임계 영역(Critical Section) :** 상호 배제 조건하에 수행되어야 하는 프로그램 영역을 의미함.

03. 운영체제의 프로세스 관리

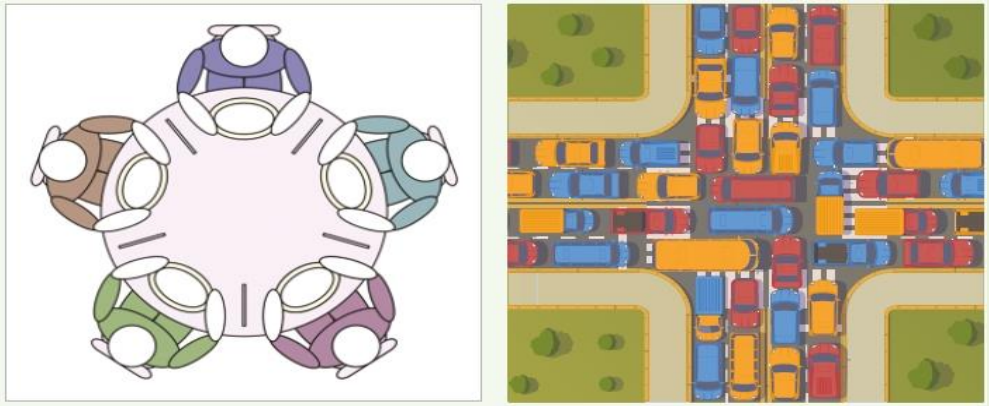
V. 교착 상태

- **교착 상태(Deadlock) :** 2개 이상의 프로세스가 절대 일어나지 않을 이벤트를 기다리고 있는 상태임.
- 특정 프로세스가 공유 자원을 독점해 사용하고 있을 때 다른 프로세스가 그 자원을 요구하면 교착 상태가 발생.

V. 교착 상태

하나 더 알기 **철학자들의 만찬 문제와 교착 상태**

- **철학자들의 만찬 문제 :** 5명의 철학자가 원탁 테이블에 놓인 젓가락을 좌우 각 1개씩 모두 2개를 가져야만 식사할 수 있는데, 모든 철학자가 동시에 왼쪽에 있는 젓가락을 잡으면 식사를 시작할 수 없음.
- **교차로에서의 교착 상태 :** 교차로의 모든 차량이 앞으로 나아가질 못하는 상황.



(a) 철학자들의 만찬 문제 (b) 교차로에서의 교착 상태

그림 4-12 철학자들의 만찬 문제와 교착 상태

03. 운영체제의 프로세스 관리

V. 교착 상태

- **자원 할당 그래프(Resource Allocation Graph) :** 노란색 원은 자원(Resource), 초록색 사각형은 프로세스(Process), 원에서 사각형으로 향하는 화살표는 자원이 그 프로세스에 이미 할당(Allocated)되었다는 뜻이며, 사각형에서 원으로 향하는 화살표는 프로세스가 해당 자원을 요청(Request)하고 있다는 뜻.
- 프로세스 P1, P2는 자원 R1, R2를 모두 할당받아야만 작업을 완료할 수 있지만, 두 프로세스 모두 자원을 하나만 할당받고 나머지 하나를 기다리는 상태. 즉, 영원히 작업을 완료할 수 없는 교착 상태에 빠진 것.

그림 4-13 자원 할당 그래프 : 교착 상태

03. 운영체제의 프로세스 관리

V. 교착 상태

- **교착 상태에 빠지는 조건**
  - 상호 배제 : 오직 하나의 프로세스만 자원을 사용할 수 있다는 조건.
  - 보유와 대기 : 자원을 할당받은 상태에서 다른 자원을 기다리는 것.
  - 비선점 : 한 프로세스가 작업을 끝내기 전까진 다른 프로세스가 해당 자원을 뺏을 수 없다는 것.
  - 순환 대기 : 각 프로세스가 자원을 가지고 있는 상태에서 다른 프로세스의 자원을 요청하는 것.

03. 운영체제의 프로세스 관리

V. 교착 상태

- **교착 상태를 해결하기 위한 운영체제의 방안**
  - 방지 : 교착 상태가 발생할 수 있는 조건 4가지 중 하나 이상을 제거해, 애초에 교착 상태가 발생하지 않도록 하는 것.
  - 회피 : 교착 상태가 발생할 가능성이 있는지 없는지 검사하고 발생할 가능성이 없을 경우에만 자원을 할당하여 발생을 회피하는 방법.
  - 탐지와 회복 : 교착 상태가 발생했을 때 이를 감지하고 상황을 해결하는 방법.
  - 무시: 교착 상태의 발생 여부를 무시하는 것.

04
운영체제의 메모리 관리

04. 운영체제의 메모리 관리

I. 메인메모리 관리 I : 단일 프로그래밍

- **단일 프로그래밍 기법 :** 운영체제 외 하나의 사용자 프로그램만 저장하는 방식으로, 한번에 오직 하나의 프로그램만 주기억장치에 저장해 실행할 수 있음.
- 사용자 프로그램을 메모리 주소의 시작 부분에 두면 운영체제는 사용자 프로그램이 운영체제로 침범하지 않도록 막기만 하면 됨.
- 운영체제의 역할이 단순해서 구현하기 쉽지만, CPU를 효율적으로 사용할 수 없음.

그림 4-14 단일 프로그래밍 관리 구조

04. 운영체제의 메모리 관리

II. 메인메모리 관리 II : 다중 프로그래밍

- **다중 프로그래밍 기법 :** 메인메모리에 여러 개의 작업을 쌓아 둔 후, CPU가 작업을 오가며 동시에 실행하는 기법.
- 다중 프로그래밍 환경의 메인메모리 관리 기법으로는 고정 분할 메모리 관리와 가변 분할 메모리 관리가 있음.

04. 운영체제의 메모리 관리

I. 메인메모리 관리 II : 다중 프로그래밍

▪ 고정 분할 메모리 관리

- **고정 분할(Fixed Partition) 메모리 관리 기법 :** 분할된 메모리 구조를 유지하다가 프로그램이 실행되면 적당한 위치를 할당하는 방식.
- 나눠진 공간은 하나의 작업이 쌓일 수 있는 일정한 크기의 기억 공간이므로, 다른 프로세스가 침범하지 않도록 보호되어야 함. 이를 위해 상한 주소와 하한 주소 값이 저장된 레지스터를 사용.

그림 4-15 고정 분할 메모리 관리 구조

04. 운영체제의 메모리 관리

II. 메인메모리 관리 II : 다중 프로그래밍

▪ 고정 분할 메모리 관리

- **단편화 :** 작업량과 분할 공간의 크기가 일치하지 않아 빈 공간이 하는 것.
  - 내부 단편화 : 작업물이 분할 공간에 들어가 있지만 공간의 크기가 커서 빈 공간이 발생하는 것.
  - 외부 단편화 : 분할 공간의 크기가 작업량보다 작아서 빈 공간이 된 것.

그림 4-16 고정 분할 메모리 관리의 단편화 현상

04. 운영체제의 메모리 관리

II. 메인메모리 관리 II : 다중 프로그래밍

▪ 가변 분할 메모리 관리

- **가변 분할(Variable Partition) 메모리 관리 기법 :** 고정 분할 메모리 관리 기법의 단점을 보완하기 위해 등장한 기법.
- 고정된 분할 공간의 경계를 없애고 작업량이 맞는 공간을 할당하고, 작업이 완료되면 빈 공간은 다시 모아 관리.
- 사용자 프로그램이 실행되면 그때그때 알맞은 크기로 공간을 분할해 할당하는 것.

04. 운영체제의 메모리 관리

II. 메인메모리 관리 II : 다중 프로그래밍

- **가변 분할 메모리 관리**

그림 4-17 가변 분할 메모리 관리 구조

- (a) : 운영체제만 메모리에 로딩되어 있고 메인메모리는 1개의 파티션으로 존재.
- (b) : 프로세스 1~4 순서에 맞게 공간을 할당하고 빈 공간은 그대로 남아 있음.
- (c) : 프로세스 3이 종료되면서 15KB 크기의 빈 공간이 추가로 발생.
- (d) : 빈 공간을 1개로 만들기 위해 프로세스들을 재배치하는 압축 과정 실행. 합쳐진 40KB 공간은 다음에 실행될 프로세스가 사용.

04. 운영체제의 메모리 관리

II. 메인메모리 관리 II : 다중 프로그래밍

▪ 가변 분할 메모리 관리

- **빈 공간에 새로운 프로그램을 할당하는 방식 3가지**
  - 최초 적합(First Fit) : 프로그램 크기보다 큰 분할 공간 중 처음 만나는 공간을 할당.
  - 최적 적합(Best Fit) : 프로그램 크기보다 큰 분할 공간 중 가장 작은 공간을 할당.
  - 최악 적합(Worst Fit) : 프로그램 크기보다 큰 분할 공간 중 가장 큰 공간을 할당.

04. 운영체제의 메모리 관리

III. 가상메모리 관리

- **가상메모리(Virtual Memory) 관리 :** 지금 당장 실행해야 하는 부분만 메인메모리에 저장하고 나머지 프로그램은 보조기억장치에 둔 채 동작하는 방법.(SWAP 사용)
- 실행할 프로그램 크기가 메인메모리보다 크거나 개수가 많으면 메인메모리 내 공간이 부족해 프로그램이 제대로 실행되지 못하는 문제를 해결하기 위해 개발.

그림 4-18 가상메모리 관리 구조

04. 운영체제의 메모리 관리

III. 가상메모리 관리

- **페이징(Paging) 기법 :** 프로그램을 일정 크기로 나누어 페이지를 만들고 페이지 단위별로 메인메모리에 올려 동작하는 방식.

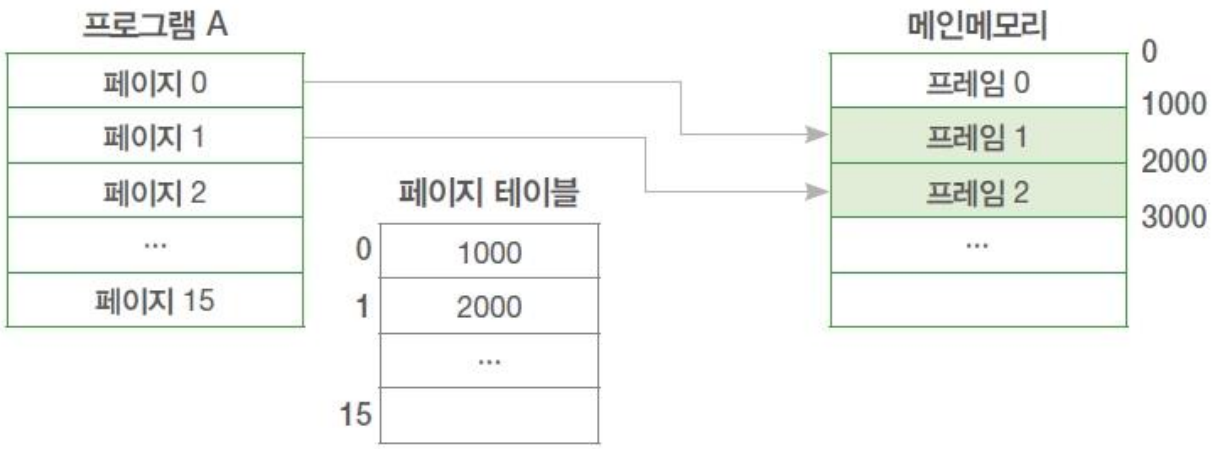


그림 4-19 페이징 기법 구조

04. 운영체제의 메모리 관리

III. 가상메모리 관리

하나 더 알기 **요구 페이징 기법**

- **요구 페이징(Demand Paging)기법 :** 기본 페이징 기법을 확장한 개념.
- CPU가 프로세스의 페이지 중 하나를 가지고 작업을 수행하다가 다른 페이지가 필요하면, 먼저 그 페이지가 메모리 내에 적재되어 있는지를 확인.
- 메모리 내에 이미 있다면 그 페이지를 사용하고, 없다면 보조기억장치에서 해당 페이지를 읽어 들임.

하나 더 알기 **세그먼테이션 기법**

- **세그먼테이션(Segmentation) 기법 :** 가상메모리를 프로그램이나 데이터 용도에 맞춰 분할하는 기법.

05. 파일시스템

I. 실행 파일과 데이터파일

- 실행 파일 : 운영체제가 메모리로 가져와 CPU를 사용하여 작업하는 파일
  (사용자 요청으로 프로세스가 되는 파일)
- 데이터 파일 : 프로세스나 응용 프로그램이 사용하는 데이터를 모아 놓은 파일

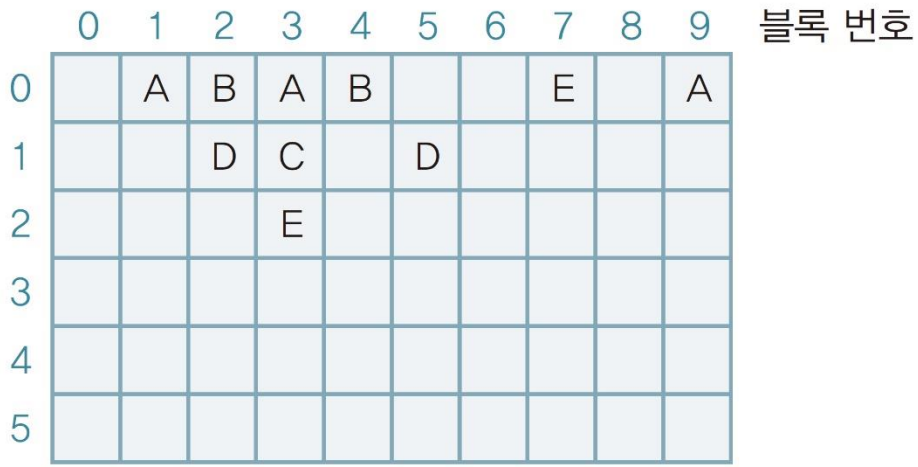
05. 파일시스템

I. 파일 테이블

- 파일이 저장된 파일 이름, 위치 정보 등이 저장
- 모든 운영체제는 고유의 파일 테이블을 가짐
- 윈도우는 FAT(File Allocation Table)나 NTFS, 유닉스는 i-node 같은 파일 시스템 운영

파일 A	1, 3, 9
파일 B	4, 2
파일 C	13
파일 D	15, 12
파일 E	23, 7

파일 테이블

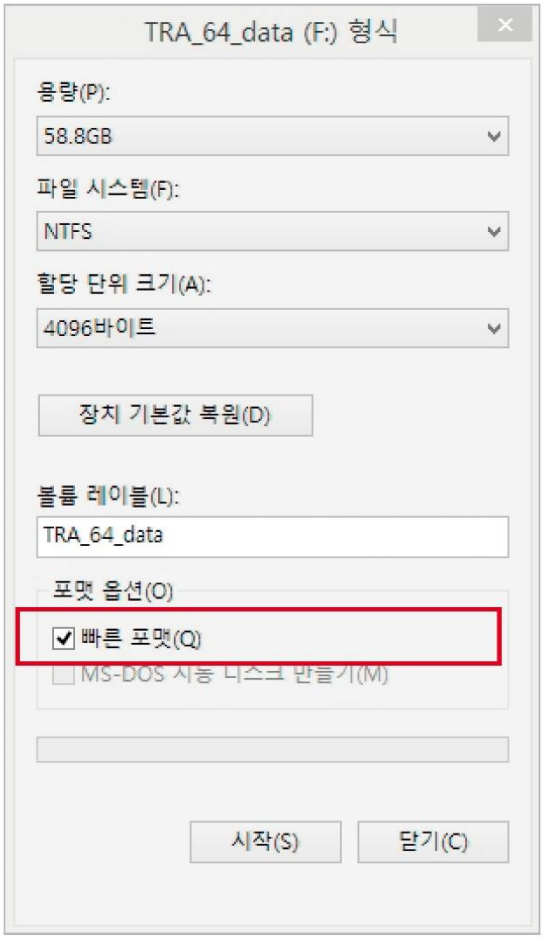


저장 장치

05. 파일시스템

I. 포맷팅(formating)

- 디스크에 파일 시스템을 탑재하고 디스크 표면을 초기화하여 사용할 수 있는 형태로 만드는 작업
- 빠른 포맷팅 : 데이터는 그대로 둔 채 파일 테이블을 초기화하는 방식
- 느린 포맷팅 : 파일 시스템을 초기화할 뿐 아니라 저장 장치의 모든 데이터를 0으로 만들어 버림



05. 파일시스템

I. 섹터

- 하드디스크의 물리적인 구조상 가장 작은 저장 단위
- 섹터마다 주소를 부여하면 너무 많은 양의 주소가 필요하기 때문에
  파일 관리자는 여러 섹터를 묶어 하나의 블록으로 만들고
  블록 하나에 주소 하나를 배정

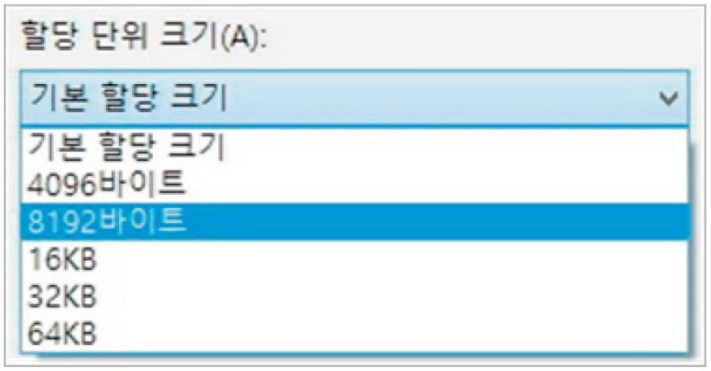
II. 블록

- 저장 장치에서 사용하는 가장 작은 단위
- 한 블록에 주소 하나를 할당
- 데이터는 운영체제와 저장 장치 간에 블록 단위로 전송

05. 파일시스템

I. 블록 크기

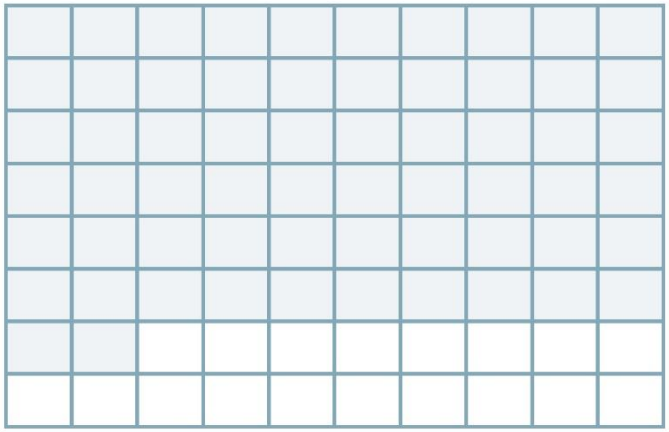
- 블록 크기는 시스템마다 다름
- 포맷할 때 시스템이 정한 기본 블록 크기를 사용 또는 4,096B~64KB의 다양한 블록 크기를 직접 지정
- 블록 크기를 작게 설정하면 저장 장치를 효율적으로 쓸 수 있지만, 파일이 여러 블록으로 나뉘어 파일 입출력 속도가 느려짐



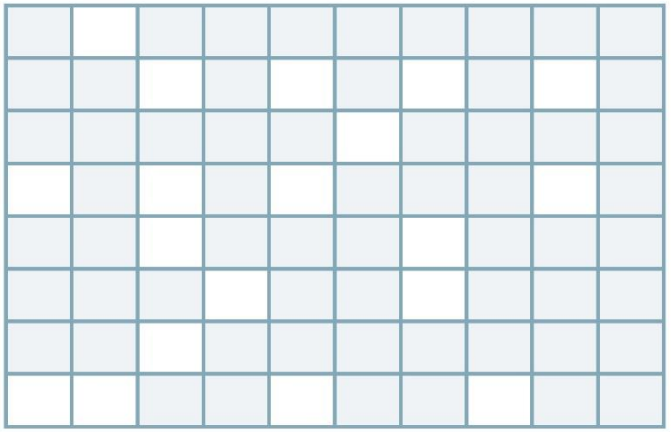
05. 파일시스템

I. 조각화(단편화)

- 하드디스크를 처음 사용할 때는 데이터가 앞부터 차곡차곡 쌓이지만, 사용하다 보면 파일이 삭제되면서 중간중간 빈 공간이 생김
- 하드디스크에 조각이 많이 생기면 큰 파일을 여러 조각으로 나누어 저장
  → 성능 저하



초기 상태



조각 난 상태

05. 파일시스템

I. 조각모음(defragmentation)

- 주기적으로 조각모음 실행
- 시간이 오래 걸리는 작업이므로 작업이 없는 특정 시간에 조각모음을 실행
- USB 메모리, SSD 등 반도체를 사용하는 저장 장치는 조각모음을 하지 않음
- 조각모음을 통해 특정 위치의 메모리만 계속 사용하면 수명 단축

05. 파일시스템

I. FAT 파일 시스템 구조

- 왼쪽 테이블은 파일 정보와 함께 파일의 시작 블록 정보를 가짐
- 파일 B : 2 → 4 → 12 → 8(null)번 블록
- 파일 C : 0 → 3 → 10 → 6 → 9 → 7 → 11 → 14(null)번 블록

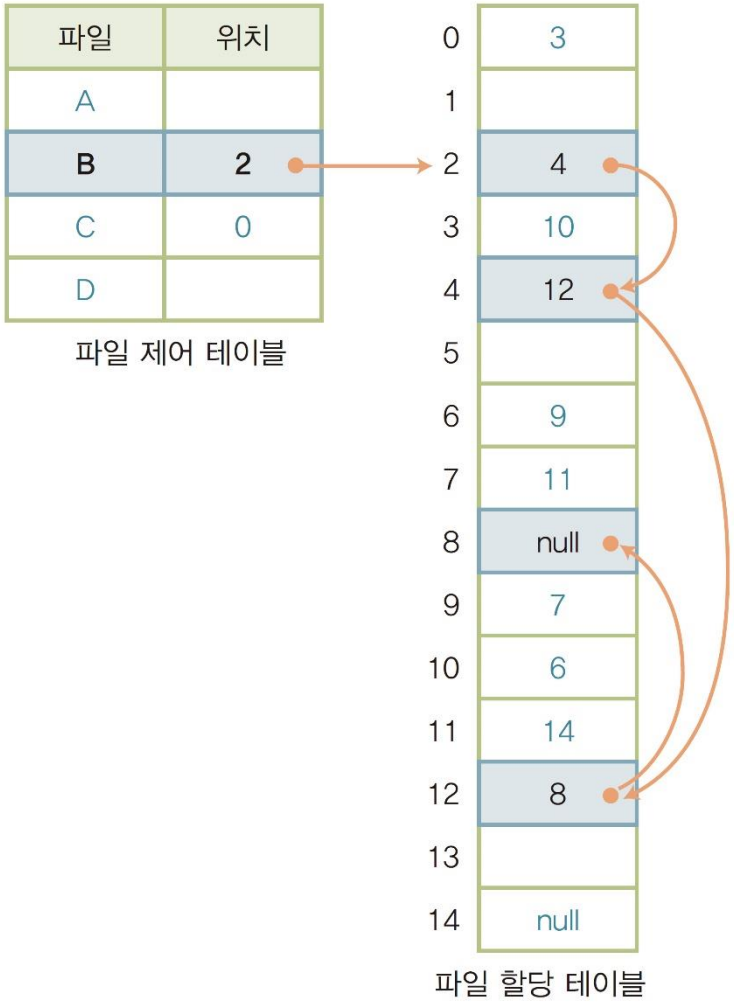


그림 6-35 FAT 파일 시스템 구조

05. 파일시스템

I. FAT32와 NTFS

- 윈도우는 파일 시스템으로 FAT32 또는 NTFS를 사용
- FAT32는 32GB까지 지원하고 파일 하나의 크기가 4GB로 한정
- USB 메모리는 대부분 FAT32를 사용
- 4GB보다 큰 파일을 저장하려고 하면 빈 공간이 있어도 '빈 공간 없음'이라는 메시지를 표시
- FAT32가 4GB 이상의 파일을 지원하지 않아 발생하는 문제
- NTFS 파일 시스템으로 바꾸면 해결
- 대신 NTFS는 자동차나 오디오와 같은 기기에서 인식이 안될 수 있음.

05. 파일시스템

I. 데이터 관리

- 파일을 지우면 파일 내용이 사라지는 것이 아니라 파일 테이블에서 파일 정보만 삭제
- 새로운 파일을 저장하면 방금 지운 파일 공간에 덮어 쓰는 것이 아닌 앞의 빈 공간부터 덮어 씀