Chapter 4 운영체제 기초 (Operating System)

2020. 1학기

설명할 내용

1 주기억장치

2 파일시스템

2.1 윈도우 파일시스템

2.2 유닉스 파일시스템

1. 주기억장치 관리

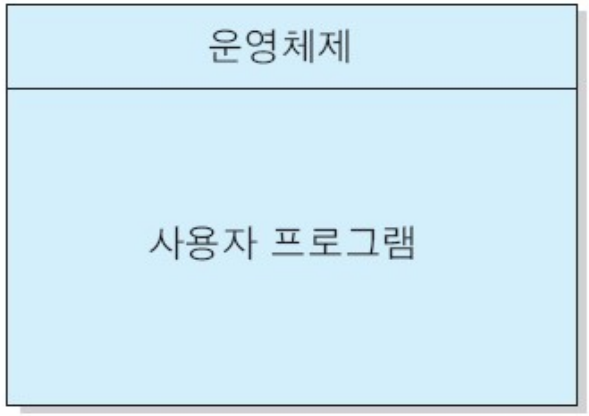
1. 단순한 구조의 주기억장치 관리

A. 단일 연속(**Single Contiguous**) 주기억장치 관리와 분할(**Partition**) 주기억장치 관리

2. 단일 연속 주기억장치 관리

A. 주기억장치에 운영체제 외에 한 개의 사용자 프로그램만 저장

B. 주기억장치를 두 영역으로 나누어 **한 영역에는 운영체제를 저장**하고 **다른 영역에는 한 개의 사용자 프로그램을 저장**



1. 주기억장치 관리

3. 분할 주기억장치 관리

A. 주기억장치를 **n개의 영역으로 분할**하여 각 영역에 서로 다른 프로세스를 동시에 저장하는 방식

B. 분할 주기억장치에서의 프로그램을 메모리에 적재할 때 메모리를 할당 방식 – **First Fit / Best Fit / Worst Fit**

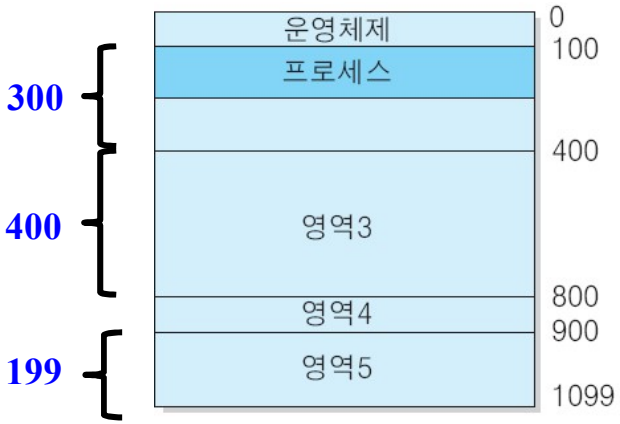
할당 방식	설명
최초 적합(first fit)	프로세스의 크기보다 큰 최초의 영역에 할당한다.
최적 적합(best fit)	프로세스의 크기보다 큰 영역 중 가장 작은 영역에 할당한다.
최악 적합(worst fit)	프로세스의 크기보다 큰 영역 중 가장 큰 영역에 할당한다.

1. 주기억장치 관리

프로세스의 크기를 150으로 가정

– 요구되는 메모리의 크기

① **최초 적합 방식**: 프로세스는 영역 2, 영역 3 그리고 영역 5 중에서 최초 영역인 영역 2에 저장된다.



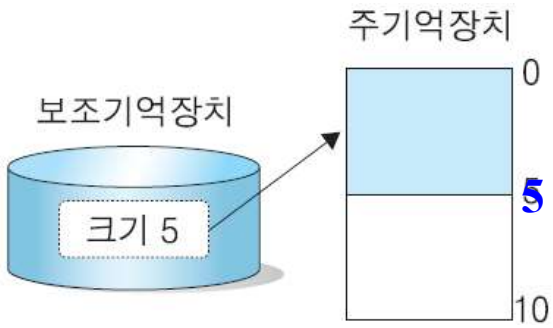
② **최적 적합 방식**: 프로세스는 영역 2, 영역 3, 영역 5 중 가장 작은 영역인 영역 5에 저장된다.

③ **최악 적합 방식**: 프로세스는 영역 2, 영역 3, 영역 5 중 가장 큰 영역인 영역 3에 저장된다.

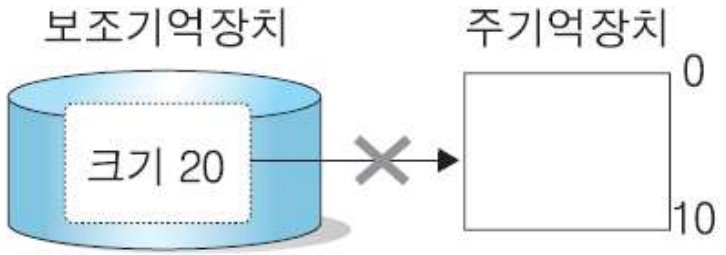
1. 주기억장치 관리: 가상메모리(Virtual Memory)

4. 가상 메모리

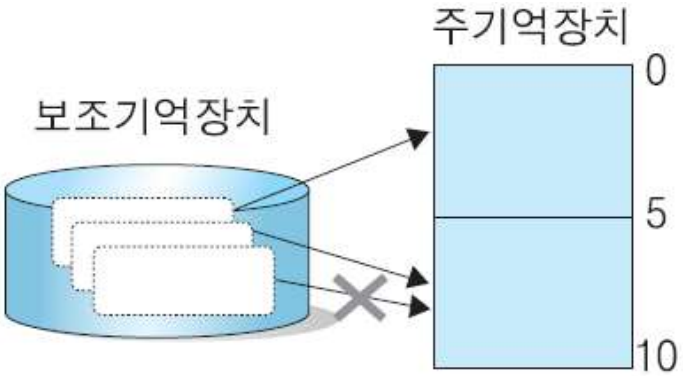
A - 크기 5의 프로그램이 주기억장치에 할당: 문제없음



B1 - 주기억장치보다 프로그램 크기가 큰 경우: **문제** 발생



B2 - 실행하려는 전체 프로그램 크기가 주기억장치보다 큰 경우: **문제** 발생



Virtual Machine, VM

– 하나의 컴퓨터에서 **2개 이상의 운영체제를 가동**하는 것.

Machine은 OS의 의미.

1. Boot시 OS 선택
2. 명령어 이용으로 OS 선택
   – 다른 Window에서는 다른 OS

예) **VirtualBox**, **Vmware**, Virtual PC

1. 주기억장치 관리: 가상메모리(Virtual memory)

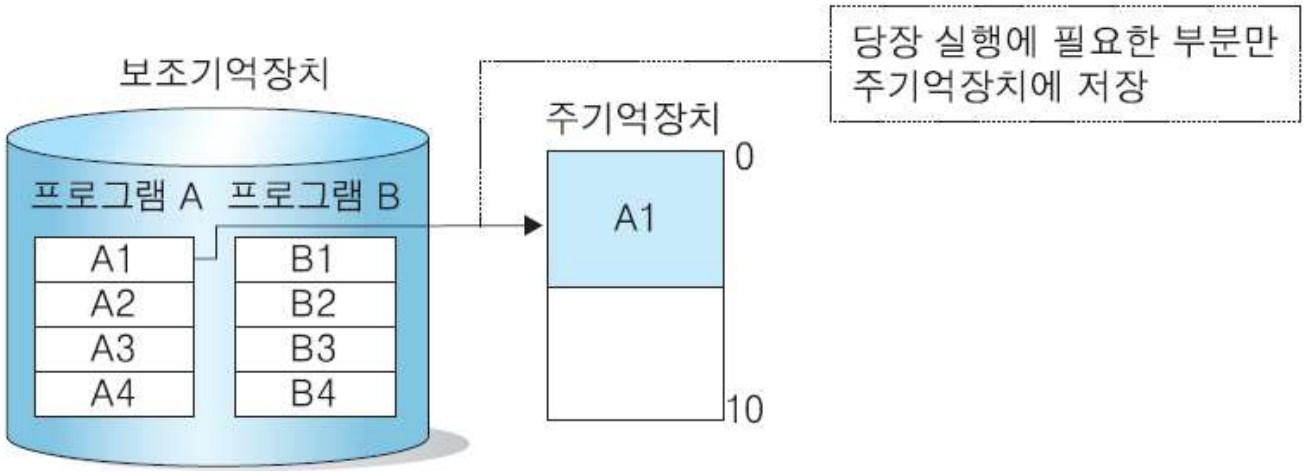
4. 가상메모리 (계속)

A. 당장 실행에 필요한 부분만 주기억장치에 저장하고 (당장 필요하지 않은) 나머지 부분은 보조기억장치에 넣어 두고 실행

B. 사용자(User)는 실제 주기억장치보다 큰 주기억장치를 가지고 있는 것처럼 느끼게 됨

C. 당장 실행될 부분만 주기억장치에 저장한다.

- **페이지(Page):** 프로그램을 일정한 크기로 나눈 단위
- 페이징(Paging): 페이지 단위로 주기억장치에 올리며 동작하는 것



1. 주기억장치 관리: 가상메모리(Virtual memory)

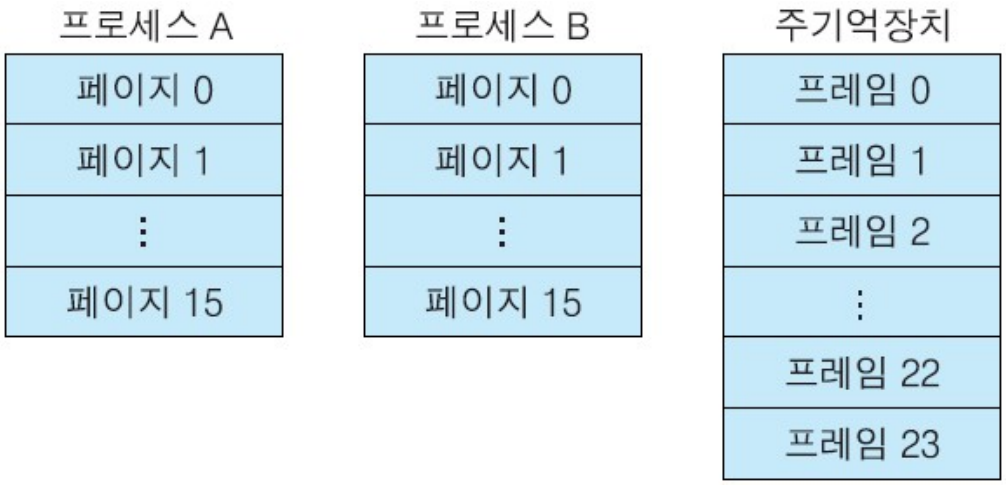
5. 가상메모리: 페이징(Paging)

A. 가상메모리를 구현하는 **한 방법**으로, 가상메모리 공간을 일정한 크기의 페이지로 나누어서 관리

- **페이지 프레임(Page Frame)**: 실제 주기억장치의 페이지에 해당하는 부분

B. 동작 원리

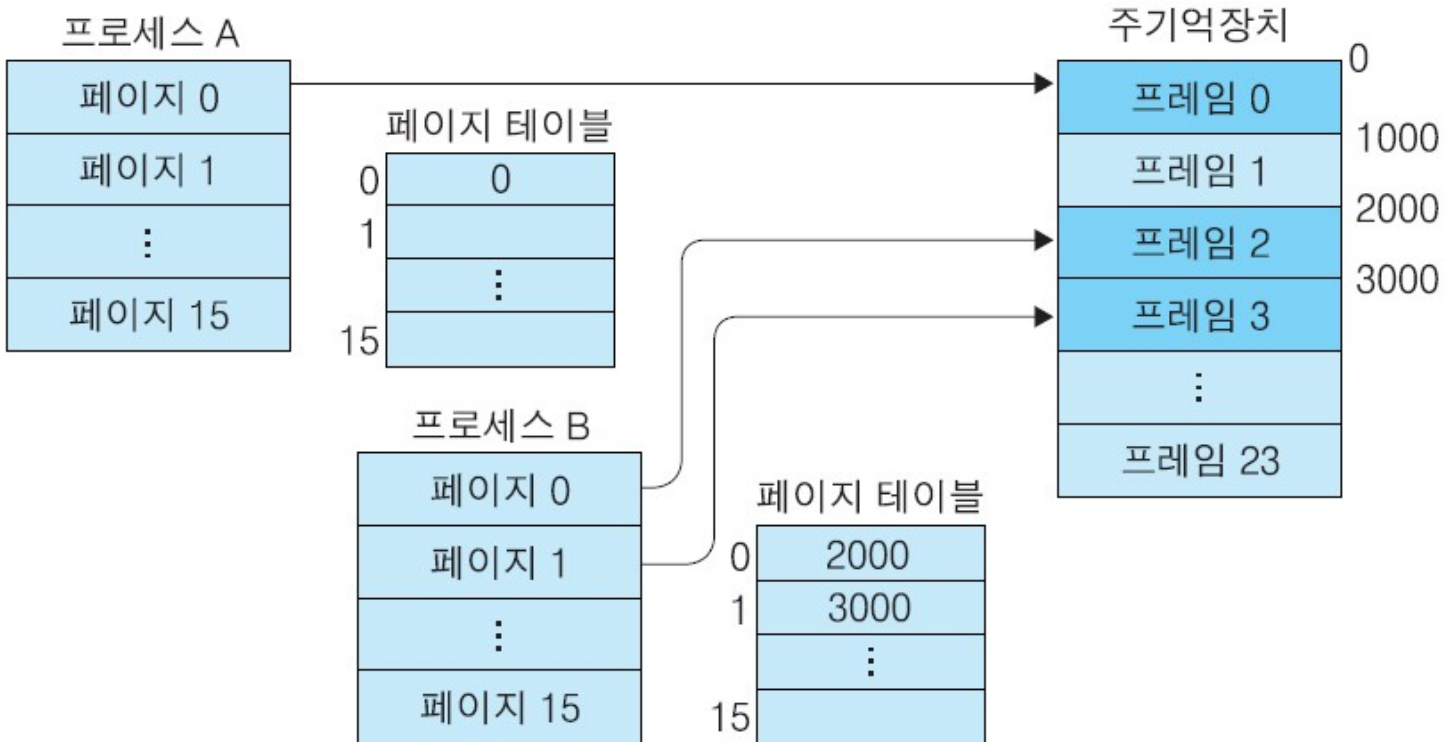
예) 16개 페이지로 이루어진 프로세스 A와 프로세스 B가 실행 중이고, 주기억장치는 24개 페이지 프레임을 가지고 있다.
(각 페이지의 크기는 1000이라 가정)



1. 주기억장치 관리: 가상메모리(Virtual memory)

C. 동작 예) - 계속

프로세스 A의 페이지 0과 프로세스 B의 페이지 0, 1이
당장 실행되어야 한다고 가정



* **페이지 테이블: 프로세스 마다 각 페이지가 주기억장치의 어느 프레임에 저장되는지를 나타내는 테이블**

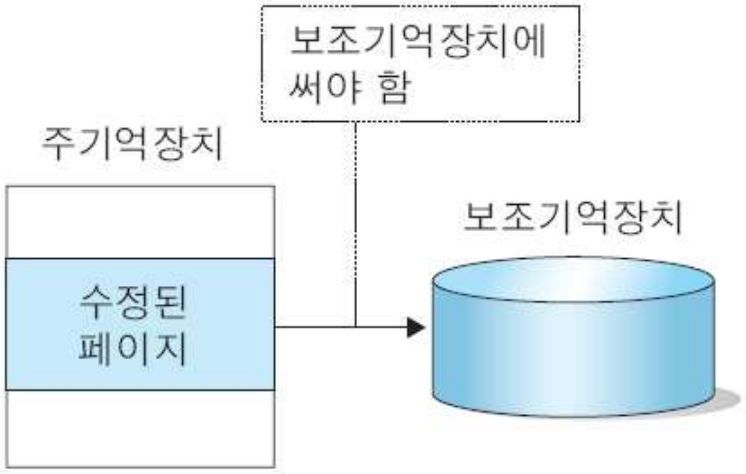
1. 주기억장치 관리: 가상메모리(Virtual memory)

D. 페이지 교체 알고리즘

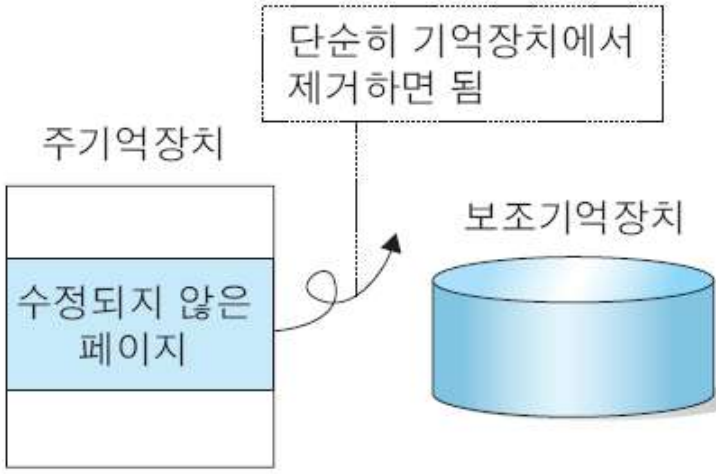
- 새로운 페이지를 주기억장치에 저장할 때 비어있는 프레임이 없으면 새로운 페이지를 저장하기 위해 주기억장치에서 제거할 페이지를 결정하는 동작

생각해 볼 수 있는 페이지 교체 방법은 ?

* 페이지 교체할 때의 동작



(a) 제거할 페이지가 수정되었을 경우



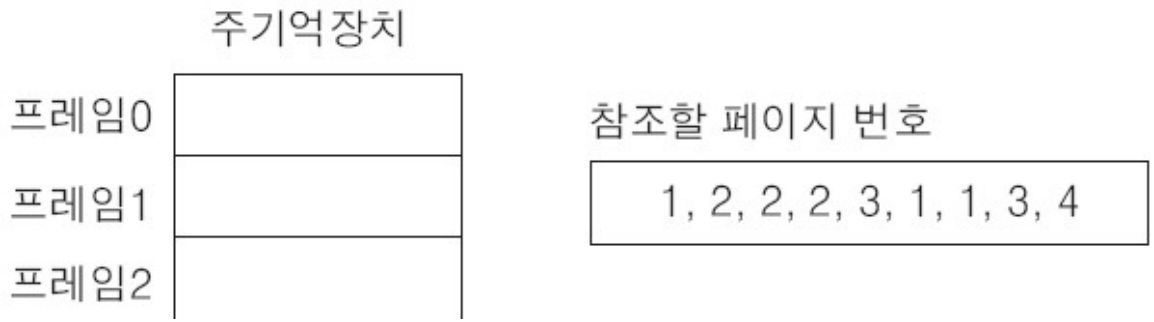
(a) 제거할 페이지가 수정되지 않았을 경우

1. 주기억장치 관리: 가상메모리(Virtual memory)

E. 페이지 교체 알고리즘의 동작 예)

가정: 페이지 프레임이 3개인 주기억장치가 있고,

참조한 페이지 번호(참조열, Reference String)가 1, 2, 2, 2, 3, 1, 1, 3, 4라 할 때의 예를 이용해서 각각의 알고리즘에 대해 살펴보자.



페이지 교체 방법(알고리즘)

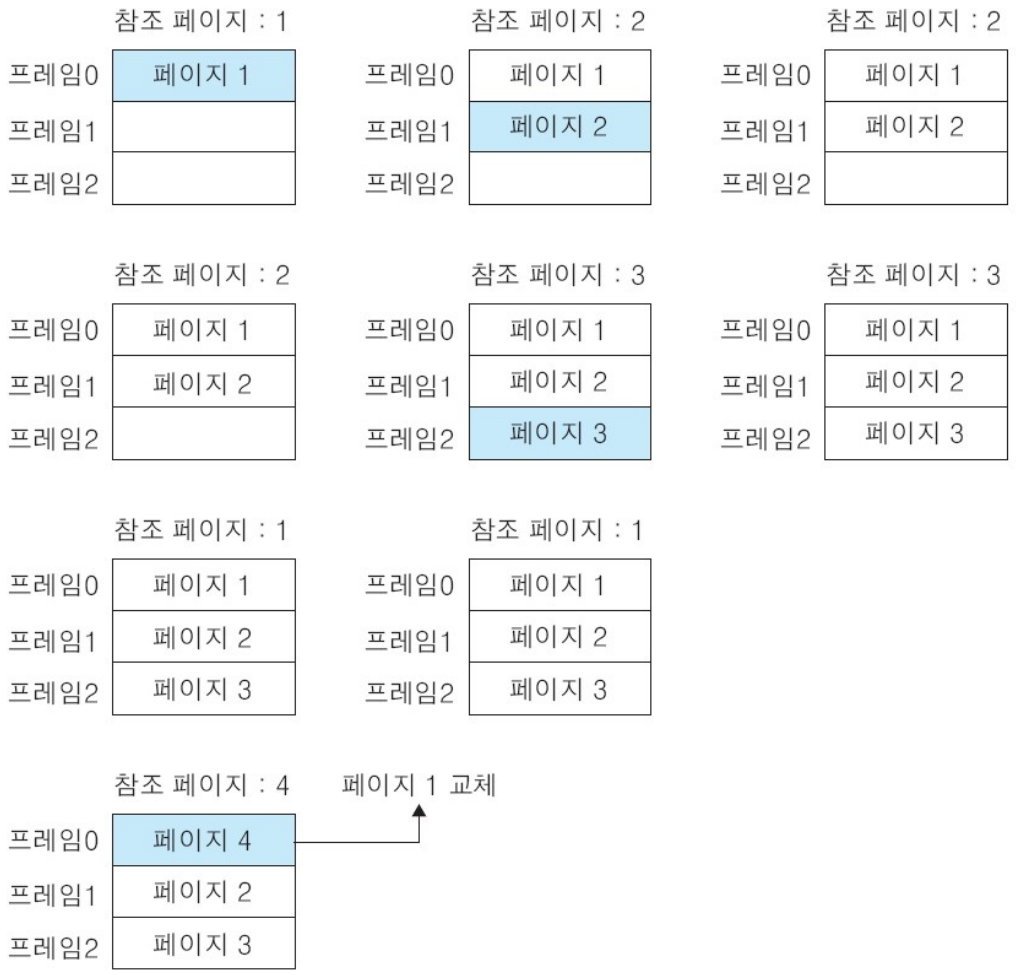
1. **FIFO(First-In First-Out)**알고리즘
2. **LRU(Least Recently Used)** 알고리즘
3. **LFU(Least Frequently Used)** 알고리즘

1. 주기억장치 관리: 가상메모리(Virtual memory)

5.1. FIFO(First-In First-Out)알고리즘

A. 페이지를 교체해야 할 때 주기억장치에 가장 먼저 올라온(Load) 페이지를 선택해서 제거하는 기법

참조한 페이지 번호
1, 2, 2, 2, 3, 1, 1, 3, 4

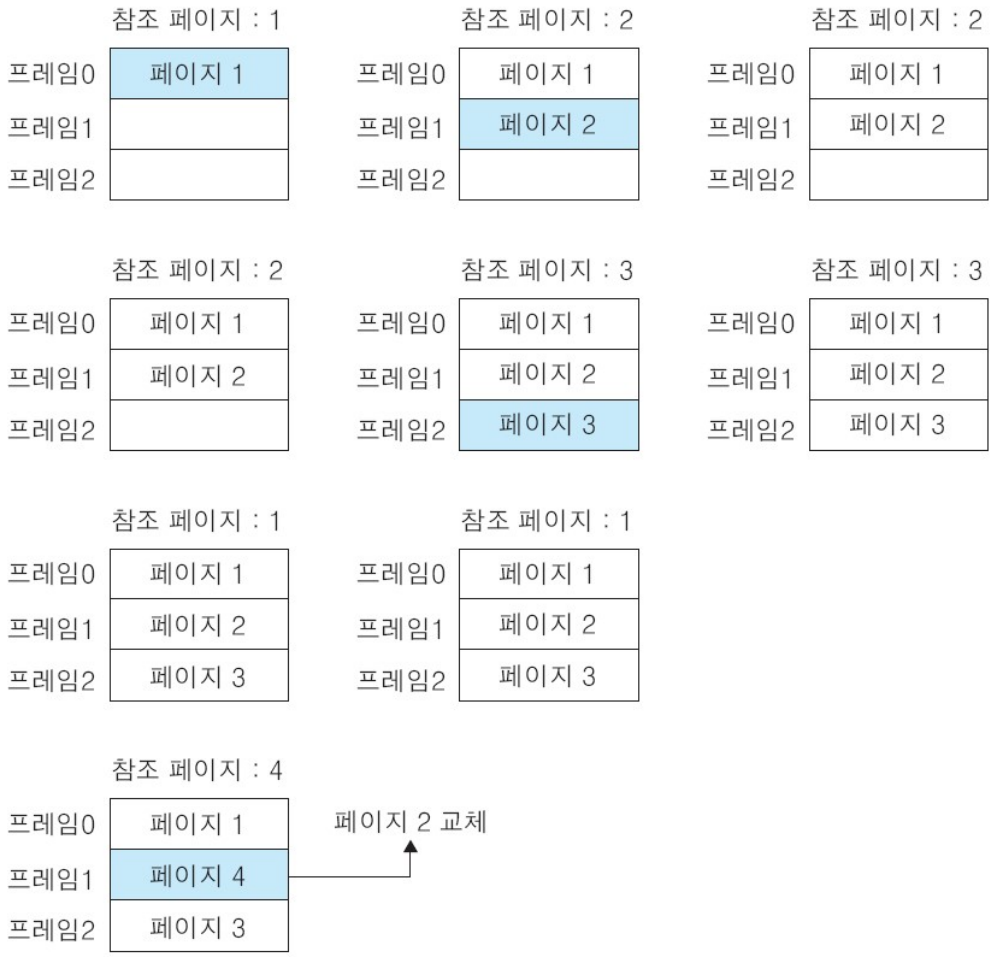


1. 주기억장치 관리: 가상메모리(Virtual memory)

5.2. LRU(Least Recently Used) 알고리즘

A. 페이지를 교체해야 할 때, 주기억장치에 올라온 페이지들 중에서 가장 오랫동안 사용되지 않았던 페이지를 제거하는 기법

참조한 페이지 번호
1, 2, 2, 2, 3, 1, 1, 3, 4



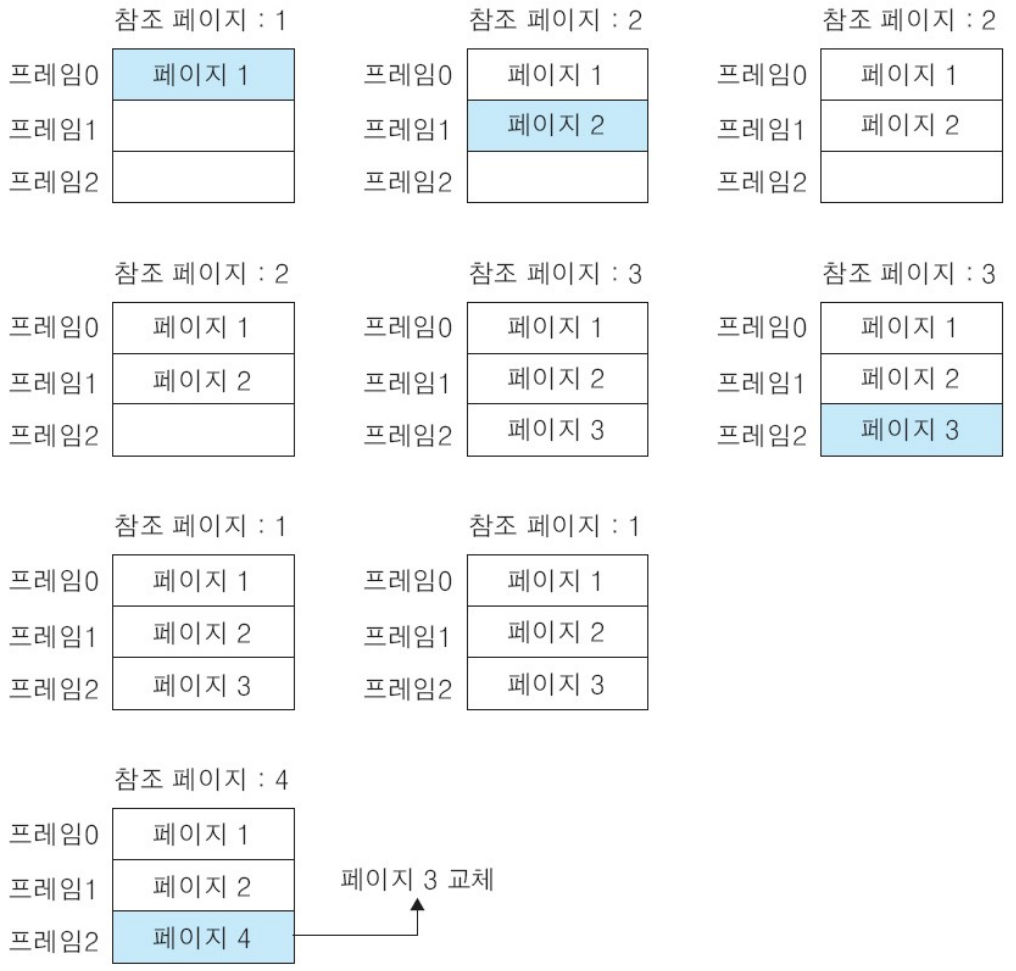
1. 주기억장치 관리: 가상메모리(Virtual memory)

5.3. LFU(Least Frequently Used) 알고리즘

A. 페이지를 교체해야 할 때 페이지들 중 사용빈도가 가장 낮은 페이지를 선택해서 제거하는 기법

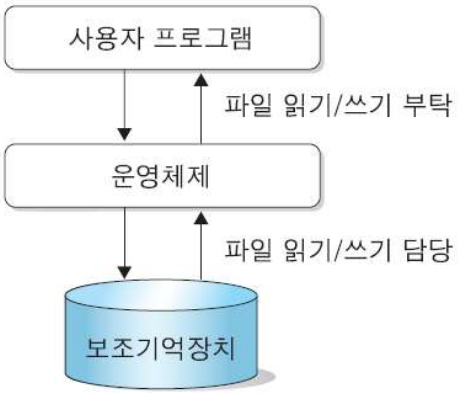
참조한 페이지 번호

1, 2, 2, 2, 3, 1, 1, 3, 4



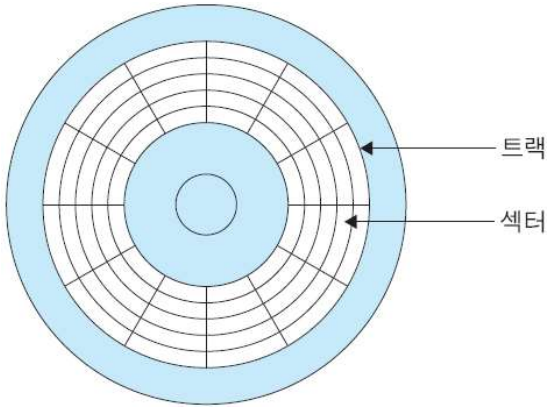
2. 파일시스템

2.1 파일 시스템 - 컴퓨터에서 자료를 쉽게 발견하고, 자료에 신속히 접근할 수 있도록 파일을 보관하거나 조직하는 체제(파일에 이름을 붙이고, 저장이나 검색을 위해 파일을 어디에 위치시킬 것인지를 나타내는 체계).



파일을 읽고 쓰는 동작은
운영체제에서 담당

윈도우의 파일시스템 (FAT 방법)
- **File Allocation Table** (FAT)
- 포맷된 디스크의 물리적 구조



2.2 Windows 파일 시스템 종류

FAT32 - File Allocation Table

FAT는 파일 할당 테이블의 약자로, FAT 형식은 메모리카드나 USB 등에 쓰인다. 이 FAT32는 스마트 폰에 OTG로 연결해서도 사용 할 수 있다. NTFS 보다 단순하기 때문에 Access 속도가 조금 빠르다. 단점은 파일의 크기 제한이 4GB 이다.

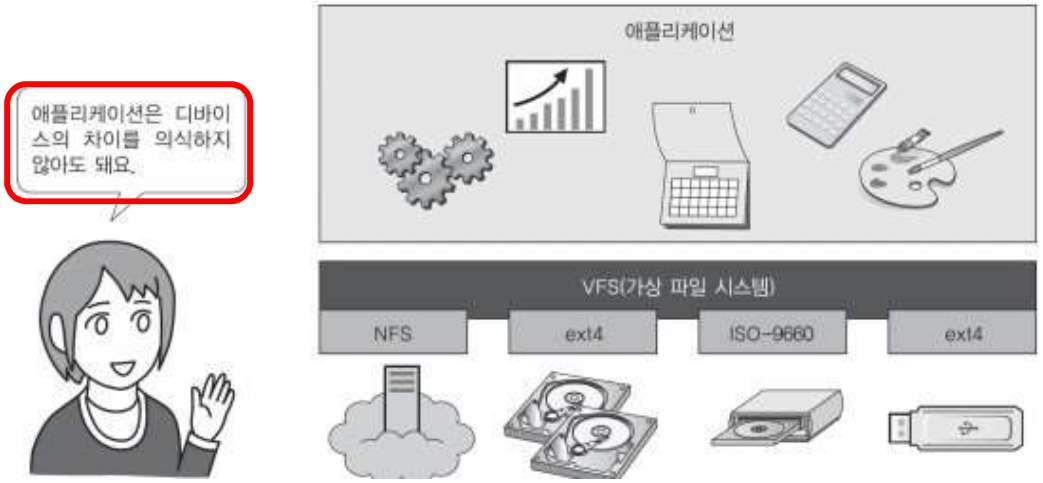
NTFS - New Technology File System

이 파일 시스템은 FAT32를 대체하기 위해 만든 파일 시스템이다. 따라서 4GB의 크기 제한을 초과하여 16TB까지 파일 크기를 지원한다. 또한 안정성과 보안성이 좋지만, Access 속도가 조금 느리다.

exFAT - Extended File Allocation Table

FAT64 라고도 불리운다. NTFS와 호환성 제공 및 FAT32의 용량 한계성을 극복한, 지원 최대 파일 크기는 512TB 이다. 안정성이 미약하다는 단점이 있다.

2.3 Linux 파일 시스템 종류



VFS – Virtual File System.
UNIX SVR4에서 추가.
유닉스에서는 표준의 파일 시스템 외에 4.3BSD의 고속 파일 시스템(FFS), NFS, RFS 그리고 파일적인 각종 디바이스가 있는데 VFS는 이들을 통합한 것.

이름	설명
ext2	Linux 운영체제에서 널리 이용되던 파일 시스템. 초기 ext 파일 시스템을 확장했기 때문에 ext2라고 이름이 붙여졌다.
ext3	Linux에서 주로 사용되는 파일 시스템. Linux 커널 2.4.16부터 사용할 수 있다.
ext4	ext3의 후속 저널링 파일 시스템. 확장 기능을 사용하지 않는 경우에 한해 ext3으로 마운트할 수 있다. 스토리지는 1EiB까지 지원하며, 파일의 단편화를 방지하는 extent file writing이라는 시스템이 도입되어 있다.
tmpfs	Unix 계열 OS에서 임시 파일을 위한 장치를 말한다. tmpfs는 파일 시스템으로 마운트되지만, 하드디스크와 같은 영구성을 갖고 있는 기억장치 대신에 메모리상에 저장할 수 있다. /tmp로 마운트되는 경우가 많으며, /tmp에 저장한 파일의 실체는 메모리상에 저장되어 있기 때문에 서버를 재시작하면 파일은 모두 사라진다.
UnionFS	여러 개의 디렉토리를 겹쳐서 하나의 디렉토리로 취급할 수 있는 파일 시스템. 쓰기 가능한 디렉토리와 읽기 전용 디렉토리를 겹침으로써, 읽기 전용 디렉토리의 내용을 가상적으로 변경할 수 있다. 변경된 내용은 쓰기 가능한 디렉토리에 저장된다. 신뢰성과 퍼포먼스를 개선한 AUFS(AnotherUnionFS)도 있다.
ISO-9660	1988년에 ISO에서 표준화된 CD-ROM의 파일 시스템. Linux뿐만 아니라 다양한 운영체제에서 똑같은 CD-ROM을 읽어 들일 수 있다.
NFS	RFC 1094, RFC 1813, RFC 3530 등으로 정의되어 Unix에서 이용하는 분산 파일 시스템 및 그 프로토콜

3. 파일시스템: 윈도우 파일시스템 (FAT 방법)

3.1. 윈도우 시스템 디스크의 논리적인 구조

위 그림은 포맷된 디스크를 논리적인 구조로, 크게 **시스템 영역과 데이터 영역**으로 구성된다. **시스템 영역은** 디스크와 데이터 영역에 대한 중요한 정보를 저장하는, 작은 크기의 영역으로, **부트 레코드, 파일 할당 테이블(FAT), 루트 디렉토리**로 세분화된다. 그리고 **데이터 영역은 일반적인 파일과 서브디렉토리를 저장하는 영역으로 클러스터 단위로 나뉘어 관리**된다. **클러스터는 몇 개의 섹터로 이루어지는**데, 윈도우 버전에 따라 차이가 있다.

3. 파일시스템: 윈도우 파일시스템 (FAT 방법)

3.2. 윈도우 시스템 디스크의 논리적인 구조 (계속)

A. **부트 레코드(Boot record)** : 컴퓨터를 처음 켰을 때 동작하는 프로그램을 저장하고 있는 영역인데, 이 프로그램은 디스크에 저장되어 있는 운영체제를 주기억장치로 올리는 역할을 한다.

B. **파일 할당 테이블(File allocation table)** : 데이터 영역의 어느 부분이 사용되고 있는지의 여부를 나타내며, 한 파일 할당 테이블에 오류가 발생했을 때 다른 파일 할당 테이블을 이용하기 위해 테이블이 두 개가 있다.

C. **디렉토리(Directory)** : 디스크에 저장된 파일들에 대한 정보를 보관하는 장소다. 특히 루트 디렉토리는 포맷을 하면 시스템 영역에 자동적으로 생성된다.

3. 파일시스템: 윈도우 파일시스템 (FAT 방법)

3.3. 동작 예제

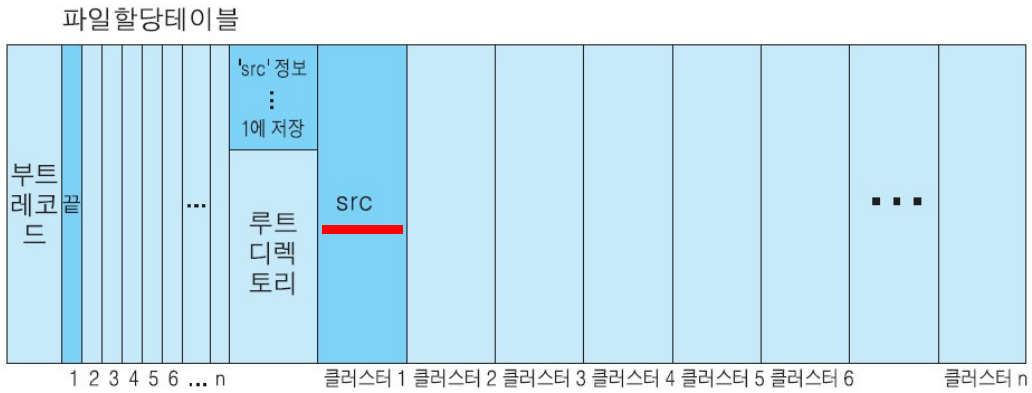
A. 클러스터 크기는 1000 byte 가정

B. /src/test.c 1500 bytes 생성

C. /c.hwp 1100 bytes 생성

D. /src/test.c 파일을 수정/변경 2500 bytes 생성

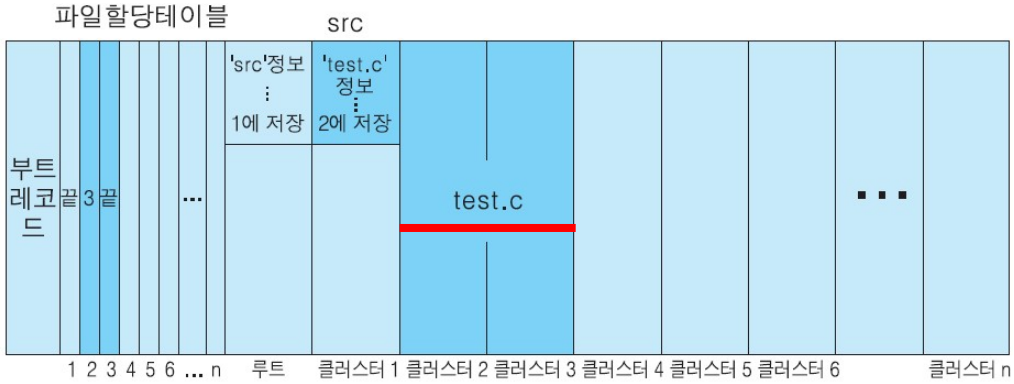
① 먼저 루트 디렉토리 아래에 src 디렉토리를 생성한다.



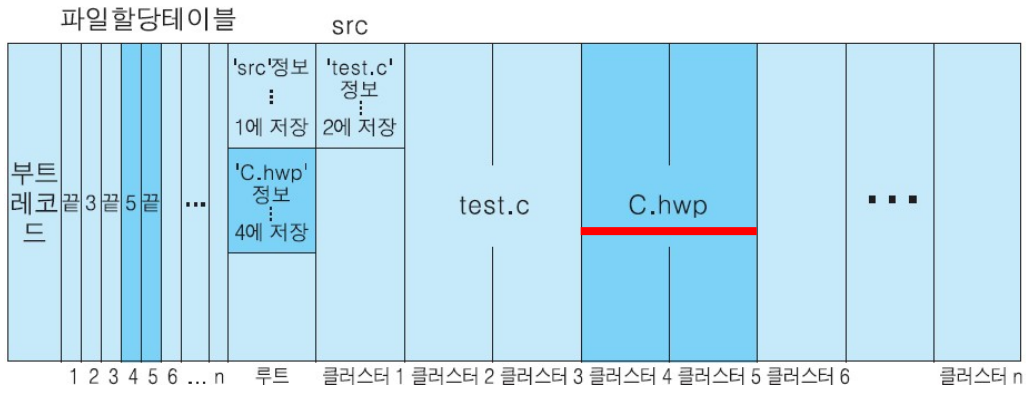
3. 파일시스템: 윈도우 파일시스템 (FAT 방법)

3.3. 동작 예제 (계속)

② src 디렉토리에 1,500바이트 크기의 test.c 파일을 생성한다.



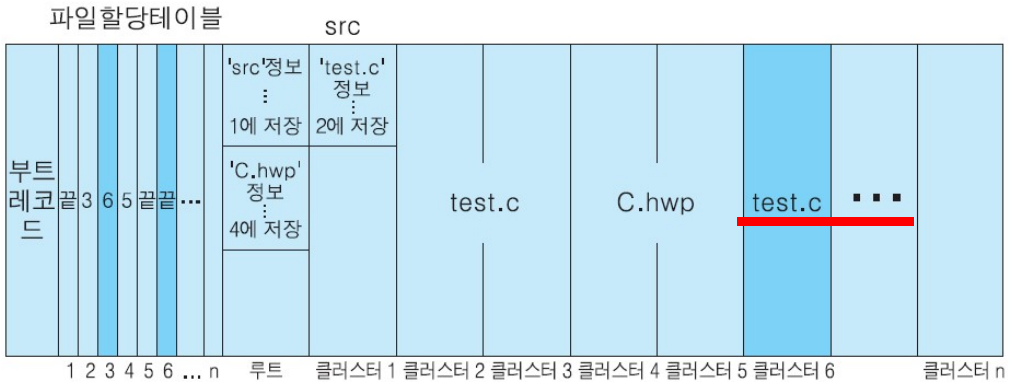
③ 루트 디렉토리에 1,100바이트 크기의 C.hwp 파일을 생성한다.



3. 파일시스템: 윈도우 파일시스템 (FAT 방법)

3.3. 동작 예제 (계속)

④ test.c 파일을 수정해서 크기가 2,500바이트로 커진다.



▸ src 디렉토리의 test.c 파일을 읽는 순서

① 루트 디렉토리에서 src 디렉토리에 대한 정보를 찾는다.

② 이 정보로 src 디렉토리가 클러스터 1에 저장된 것을 알고 클러스터1로 간다.

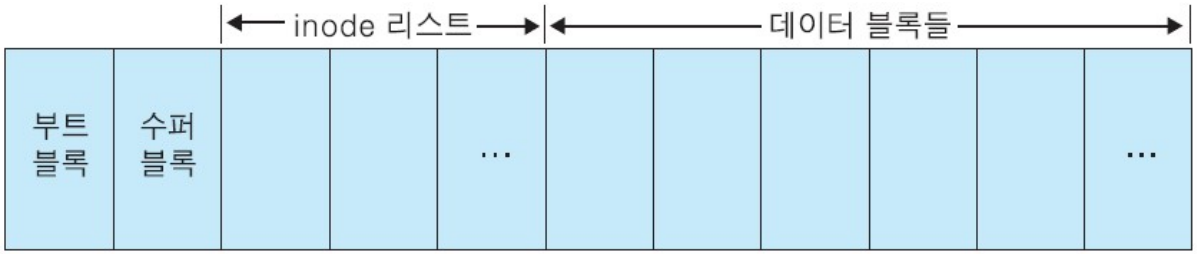
③ 클러스터 1에 위치한 src 디렉토리에서 test.c 파일에 대한 정보를 찾는다.

④ test.c 파일의 첫 부분이 클러스터 2에 저장된 것을 확인한다.

⑤ FAT 2에 3이, FAT 3에 6이 그리고 FAT 6에 끝을 의미하는 값이 저장된 것을 확인한다. 이를 통해 test.c 파일이 클러스터 2, 3, 6에 저장된 것을 알고 이들 클러스터에 저장된 내용을 읽는다.

4. 파일시스템: 유닉스(리눅스) 파일시스템

4.1. 유닉스 시스템 디스크의 논리적인 구조



A. **부트 블록**: 운영체제를 주기억장치에 올리는 역할을 하는 프로그램이 들어 있는 영역으로, 윈도우의 부트 레코드와 유사하다.

B. **수퍼 블록**: 디스크에 대한 다양한 정보를 저장한다 (전체 블록수, 블록 크기, 사용 중인 블록 수, 사용할 수 있는 블록 번호, i-node 리스트 크기, 사용할 수 있는 i-node 번호 등)

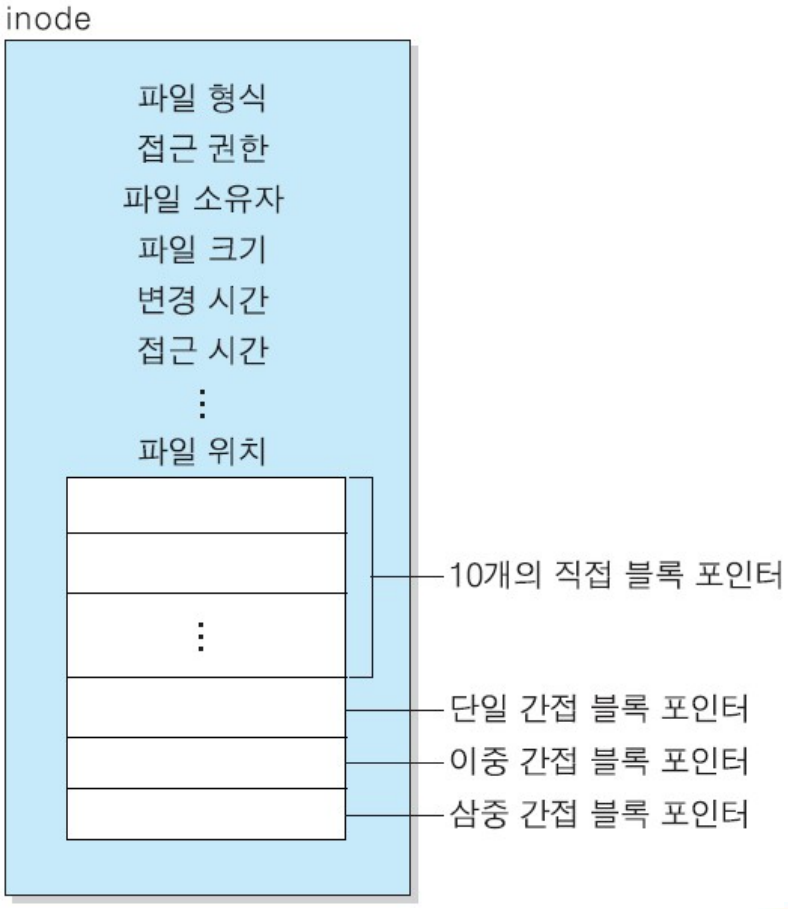
C. **i-node 리스트**: inode들을 모아놓은 곳인데, 한 블록에 여러 개의 inode를 저장하고 있다.

D. 데이터 블록: 일반적인 파일과 디렉토리 그리고 간접 블록을 저장하는 영역이다.

4. 파일시스템: 유닉스(리눅스) 파일시스템

4.2. I-node(Index Node)

A. 파일에 대한 다양한 정보(형식, 권한, 크기, 시간, 위치, ...)를 저장 (각 파일마다 하나씩 지정)

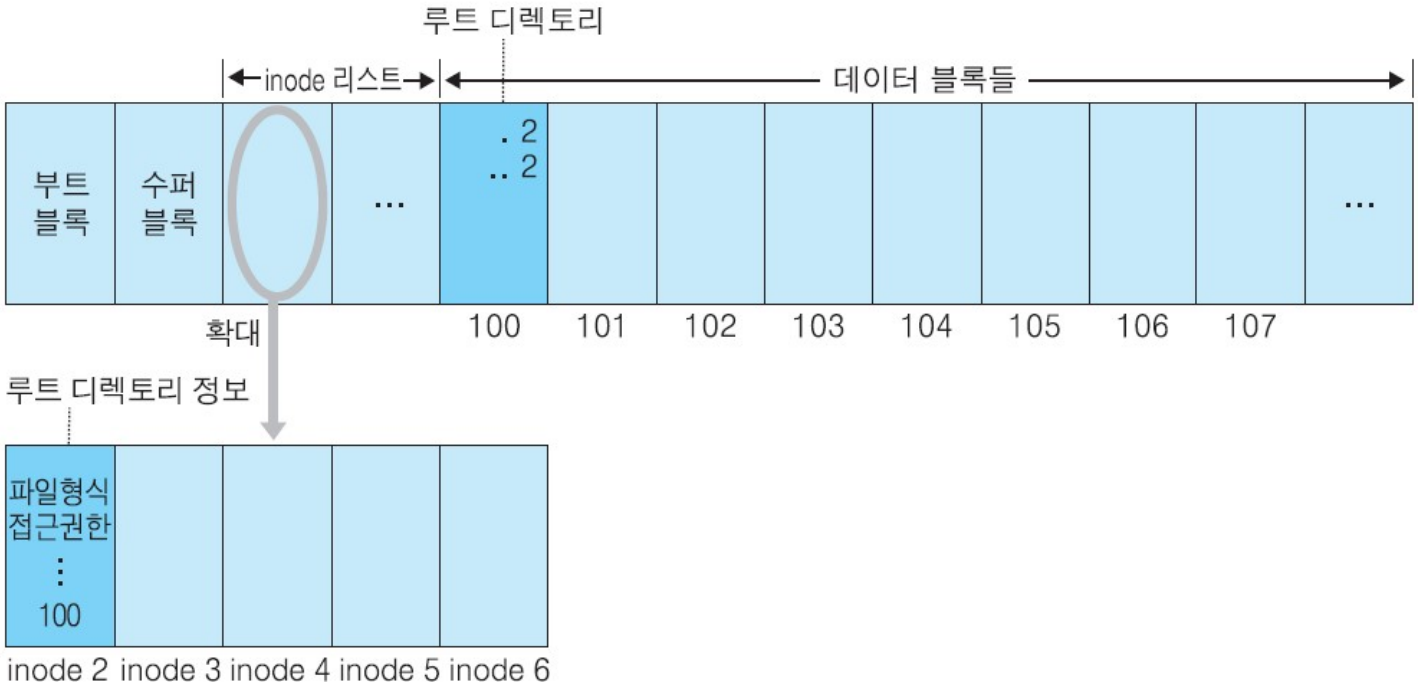


4. 파일시스템: 유닉스(리눅스) 파일시스템

4.3. 동작 예제 (블록 크기는 1000바이트 가정)

A. 윈도우의 파일 정보를 디렉토리에 저장하는 것과는 차이가 있음

B. 유닉스로 포맷한 디스크의 초기 구조

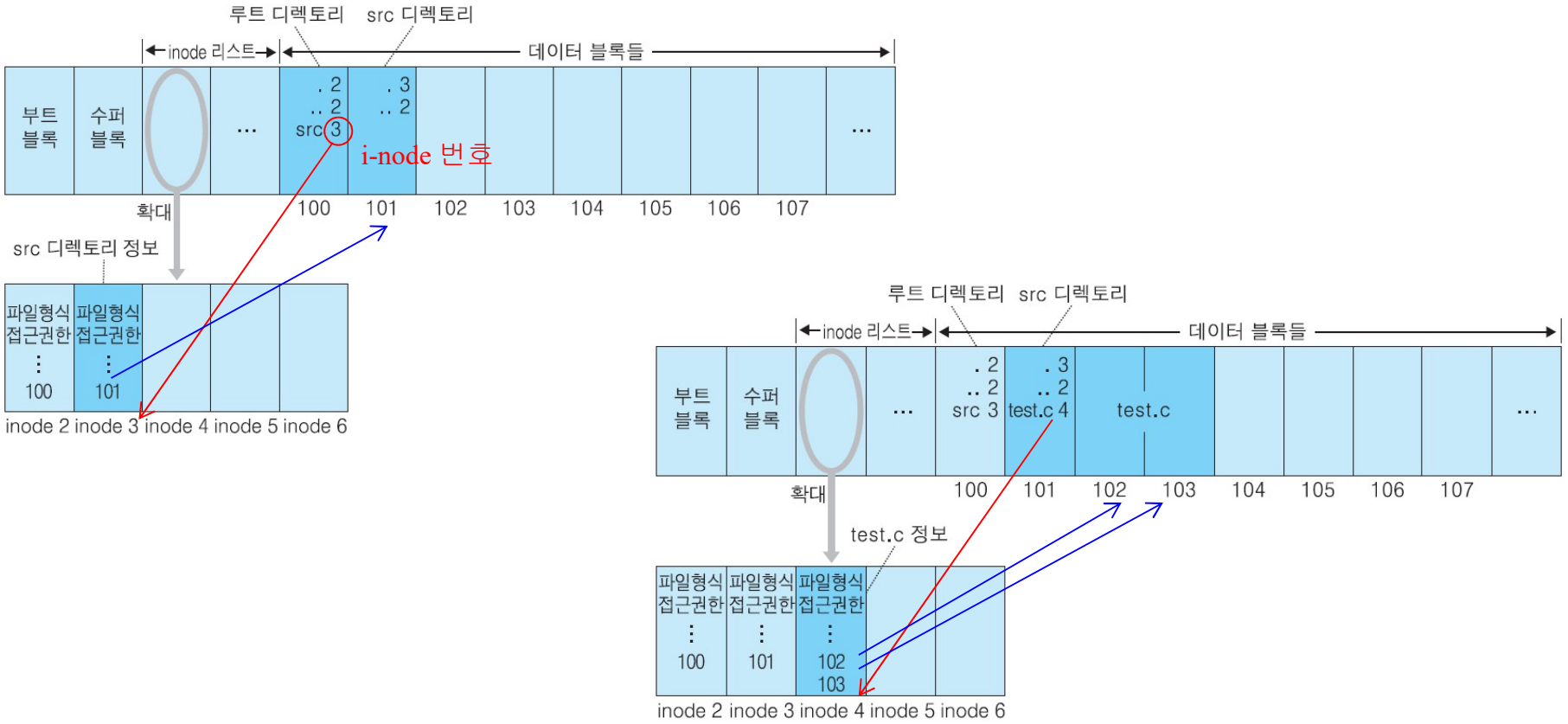


4. 파일시스템: 유닉스(리눅스) 파일시스템

4.3. 동작 예제 (계속)

① 루트 디렉토리 아래에 src 디렉토리를 생성한다.

② src 디렉토리에 1,500바이트 크기의 test.c 파일을 생성한다.

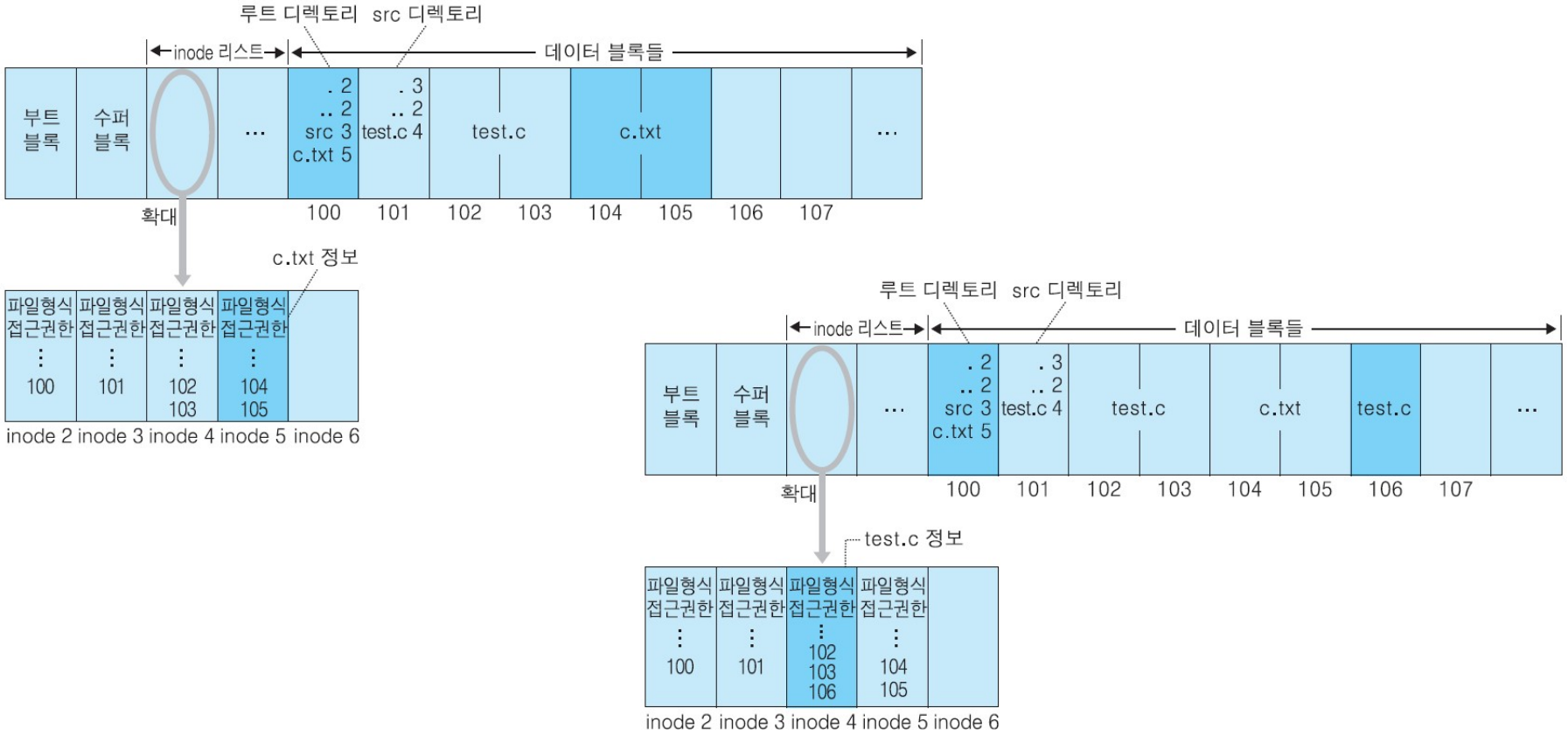


4. 파일시스템: 유닉스(리눅스) 파일시스템

4.3. 동작 예제 (계속)

③ 루트 디렉토리에 1,100바이트 크기의 c.txt 파일을 생성한다.

④ test.c 파일을 수정해서 크기가 2,500바이트로 커졌다고 하자.



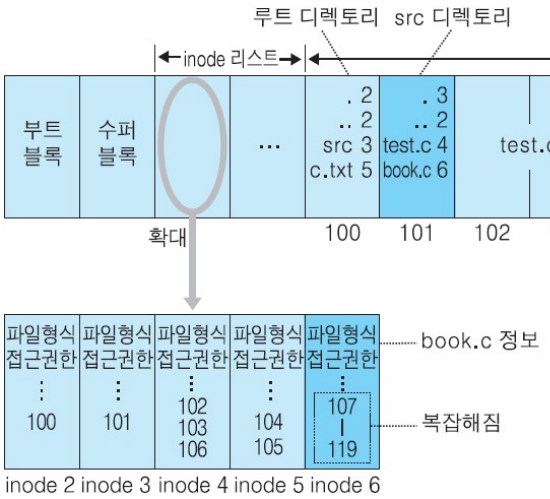
4. 파일시스템: 유닉스(리눅스) 파일시스템

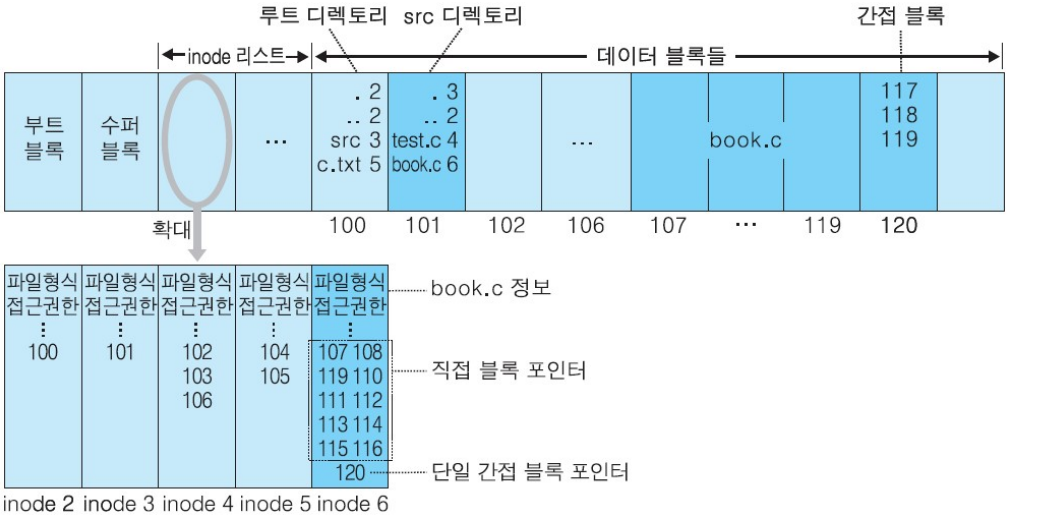
4.3. 동작 예제 (계속)

⑤ src 디렉토리에 12,500바이트 크기의 book.c 파일을 생성한다.

⑥ 간접 블록 포인터를 활용한다.

(i-node의 파일 위치를 지정하는 방법)

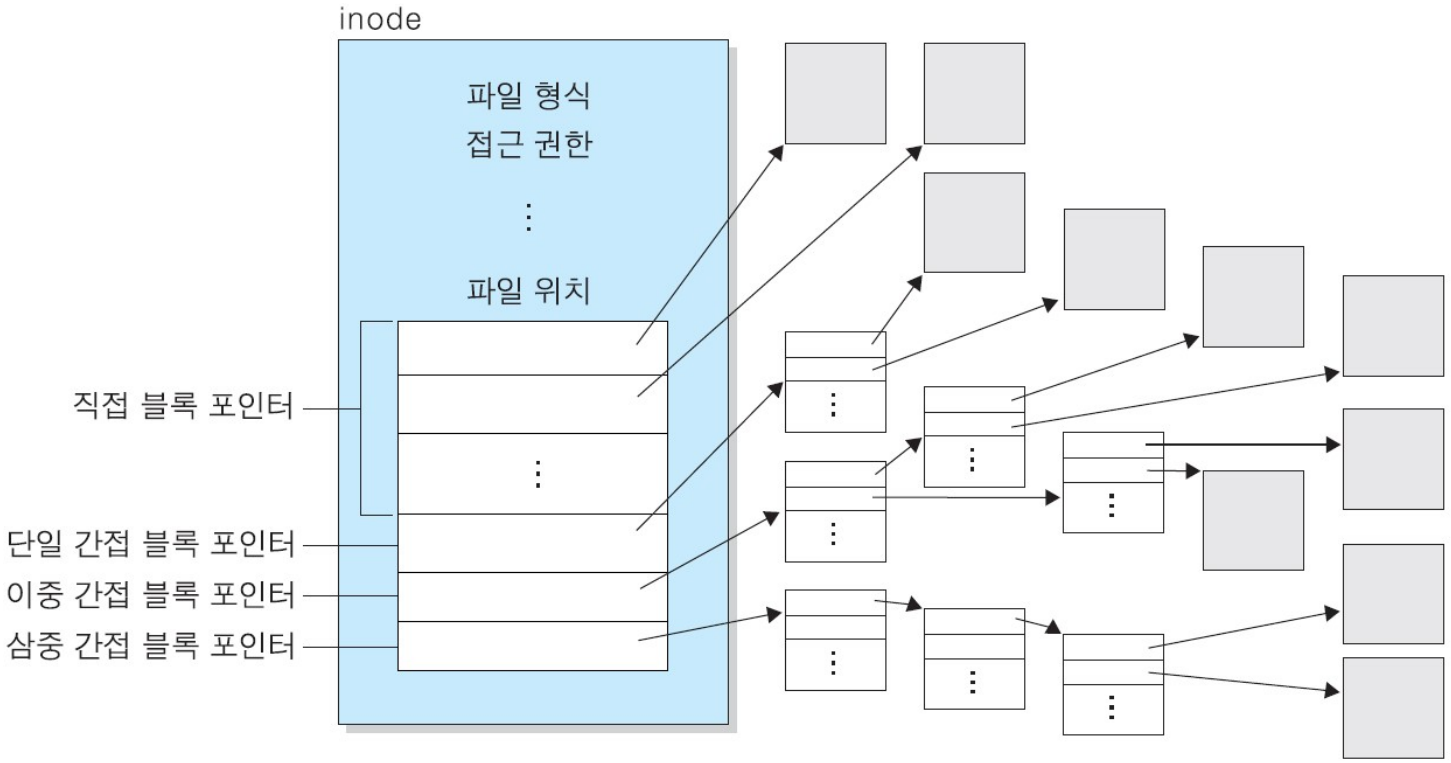




4. 파일시스템: 유닉스(리눅스) 파일시스템

4.4. 간접 블록 포인터

A. **정해진 블록 크기를 넘는 파일**인 경우를 대비한 방법



4. 파일시스템: 유닉스(리눅스) 파일시스템

4.5. src 디렉토리의 test.c 파일을 읽는 순서

① 루트 디렉토리에 대한 정보를 저장하고 있는 inode 2에서 루트 디렉토리가 블록 100에 저장된 것을 확인하고 블록 100으로 이동

② 블록 100에 위치한 루트 디렉토리에서 src의 정보가 inode 3에 저장된 것 을 알고, inode 3으로 이동. 만약 루트 디렉토리에 src에 대한 내용이 없으면 더 이상 진행하지 않고 종료

③ inode 3에서 src 디렉토리가 블록 101에 저장된 것을 확인하고 블록 101로 이동

④ src 디렉토리에서 test.c의 정보가 inode 4에 저장된 것을 알고 inode 4로 이동

⑤ inode 4에서 test.c가 블록 102, 103, 106에 저장되어 있음을 알고 이들 블록에 저장된 내용을 읽음