빅 데 이 터 혁 신 공 유 대 학

리눅스 시스템

숙명여자대학교 소프트웨어학부
창병모 교수

교육부 NRF 한국연구재단

12장 파일 시스템과 파일 입출력

01 파일 시스템

02 파일 상태 정보와 i-노드

03 디렉터리

04 링크의 구현

05 파일 입출력

빅데이터 혁신공유대학 | 교육부 NRF 한국연구재단

12.1 파일 시스템

파일 시스템 보기

- 사용법

  여러 파일 시스템 중에서 모든 파일시스템 선택.

  $ df 파일시스템*

  파일 시스템에 대한 디스크 사용 정보를 보여준다.

- 사용 예

  $ df

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
udev	1479264	0	1479264	0%	/dev
tmpfs	302400	1684	300716	1%	/run
/dev/sda5	204856328	14082764	180297788	8%	/
/dev/sda1	523248	4	523244	1%	/boot

  …

  - / 루트 파일 시스템 현재 8% 사용
  - /dev 각종 디바이스 파일들을 위한 파일 시스템
  - /boot 리눅스 커널의 메모리 이미지와 부팅을 위한 파일 시스템

디스크 사용량 보기

- 사용법

  $ du [-s] 파일명*

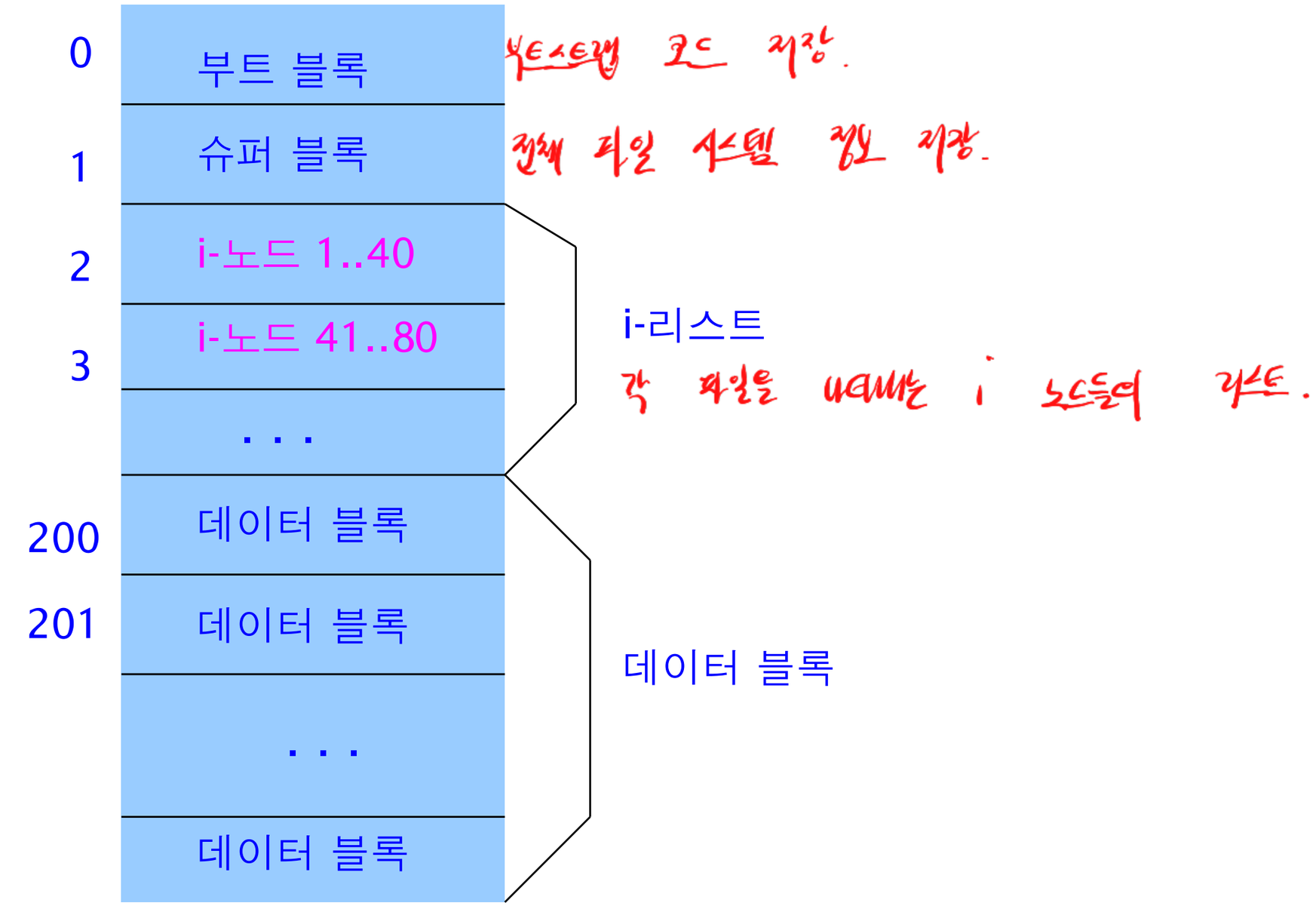
  파일 혹은 디렉터리의 사용량을 보여준다. 파일을 명시하지 않으면 현재 디렉터리 내의 모든 파일들의 사용 공간을 보여준다.

- 예

```
$ du
208 ./사진
4    ./.local/share/nautilus/scripts
8    ./.local/share/nautilus
144 ./.local/share/gvfs-metadata
4    ./.local/share/icc
...
```

```
$ du -s
22164 .
```

파일 시스템 구조


0
부트 블록
부트스트랩 코드 저장.
1
슈퍼 블록
전체 파일 시스템 정보 저장.
2
i-노드 1..40
3
i-노드 41..80
. . .
i-리스트
각 파일을 대응하는 i 노드들의 리스트.
200
데이터 블록
201
데이터 블록
. . .
데이터 블록
데이터 블록

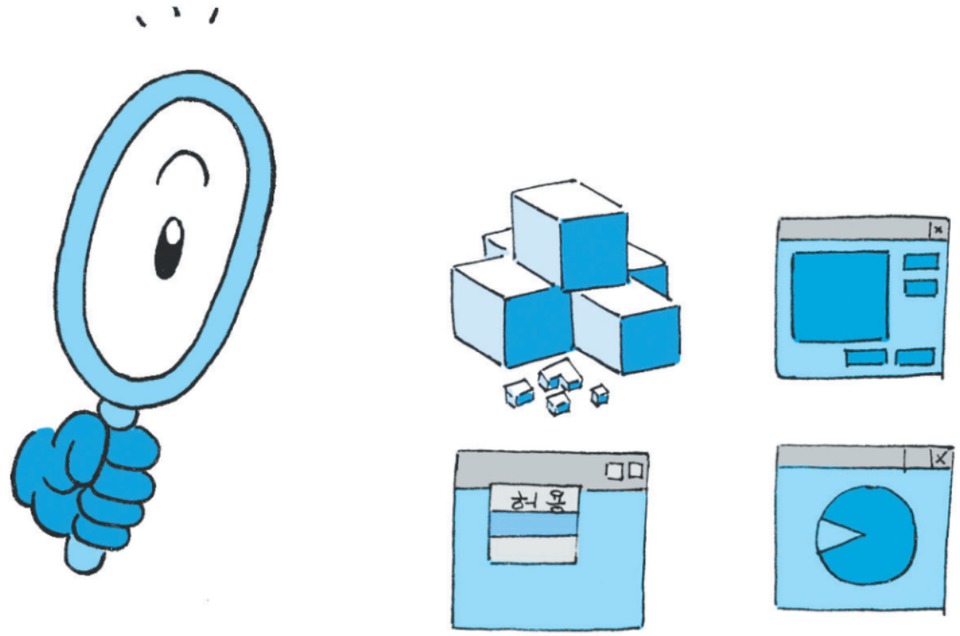
파일 시스템 구조

- 부트 블록(Boot block)
  - 파일 시스템 시작부에 위치하고 보통 첫 번째 섹터를 차지
  - 부트스트랩 코드가 저장되는 블록
- 슈퍼 블록(Super block)
  - 전체 파일 시스템에 대한 정보를 저장
    - 총 블록 수, 사용 가능한 i-노드 개수, 사용 가능한 블록 비트 맵, 블록의 크기, 사용 중인 블록 수, 사용 가능한 블록 수 등
- i-리스트(i-list)
  - 각 파일을 나타내는 모든 i-노드들의 리스트
  - 한 블록은 약 40개 정도의 i-노드를 포함
- 데이터 블록(Data block)
  - 파일의 내용(데이터)을 저장하기 위한 블록들

12.2 파일 상태 정보와 i-노드

파일 상태(file status) ⟶ i노드 내에 다 들어있다.

- 파일 상태
  - 파일에 대한 모든 정보
  - 블록 수, 파일 타입, 접근권한,
  - 링크 수, 파일 소유자의 사용자 ID,
  - 그룹 ID, 파일 크기, 최종 수정 시간, 등



- 예

```
$ ls -l cs1.txt
     4 -rw-rw-r-- 1     chang chang 2088      10월 23 13:37    cs1.txt
블록수  접근권한  링크수 사용자ID   그룹ID  파일 크기  최종 수정 시간       파일이름
```

파일 타입

stat 명령어

- 사용법

> $ stat [옵션] 파일
>
> 파일의 자세한 상태 정보를 출력한다.

- 예

```
$ stat cs1.txt
File: cs1.txt
Size: 2088      Blocks: 8          IO Block: 4096   일반 파일
Device: 803h/2051d      Inode: 1196554     Links: 1
Access: (0600/-rw-rw-r--)  Uid: (1000/chang)   Gid: (1000/chang)
Access: 2021-10-04 01:28:01.726822341 -0700
Modify: 2021-10-04 01:28:01.726822341 -0700
Change: 2021-10-04 01:28:01.726822341 -0700
 Birth: 2021-10-04 01:28:01.726822341 -0700
```

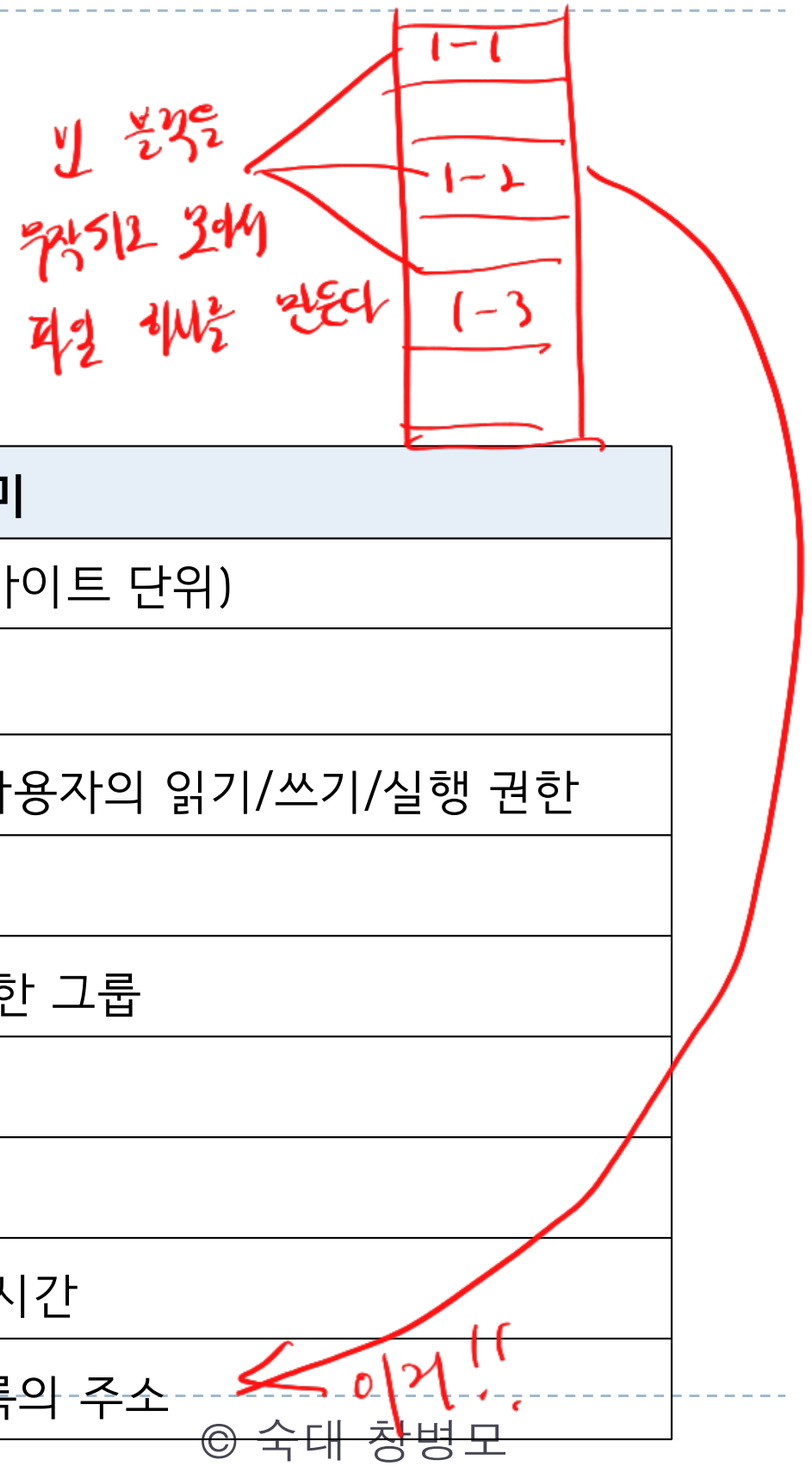
i-노드

- 한 파일은 하나의 i-노드를 갖는다.

  $ ls -i cs1.txt

  1196554 cs1.txt

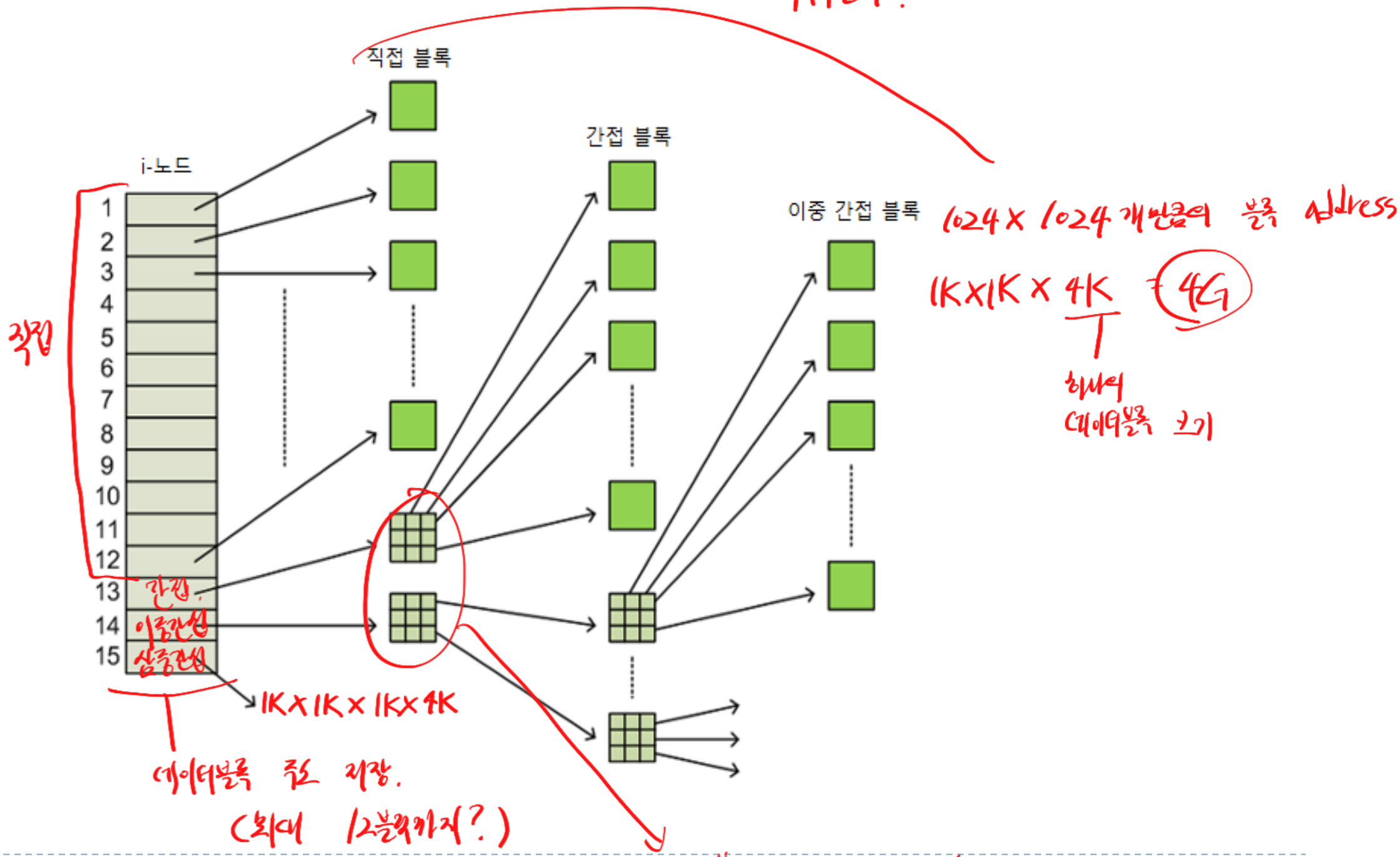
- 파일에 대한 모든 정보를 가지고 있음



파일 상태 정보	의미
블록 수	파일을 구성하는 블록의 개수(K 바이트 단위)
파일 종류	파일 종류를 나타낸다.
접근권한	파일에 대한 소유자, 그룹, 기타 사용자의 읽기/쓰기/실행 권한
하드 링크 수	파일에 대한 하드 링크 개수
소유자 및 그룹	파일의 소유자 ID 및 소유자가 속한 그룹
파일 크기	파일의 크기(바이트 단위)
최종 접근 시간	파일을 최후로 접근한 시간
최종 수정 시간	파일을 생성 혹은 최후로 수정한 시간
데이터 블록 주소	실제 데이터가 저장된 데이터 블록의 주소

© 숙대 창병모

i-노드와 블록 포인터


실제 데이터블록.
직접 블록
간접 블록
이중 간접 블록
i-노드
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
직접
간접
이중간접
삼중간접
1K×1K×1K×4K
데이터블록 주소 저장.
(최대 12블록까지?)
1024×1024개만큼의 블록 address
1K×1K×4K = 4G
해서
데이터블록 크기
12블록 이상 파일의 경우는? → 13번째부터 ~ 데이터블록의 주소들 저장.
4K/4 = 1024개 저장?

블록 포인터

- 데이터 블록에 대한 포인터
  - 파일의 내용을 저장하기 위해 할당된 데이터 블록의 주소

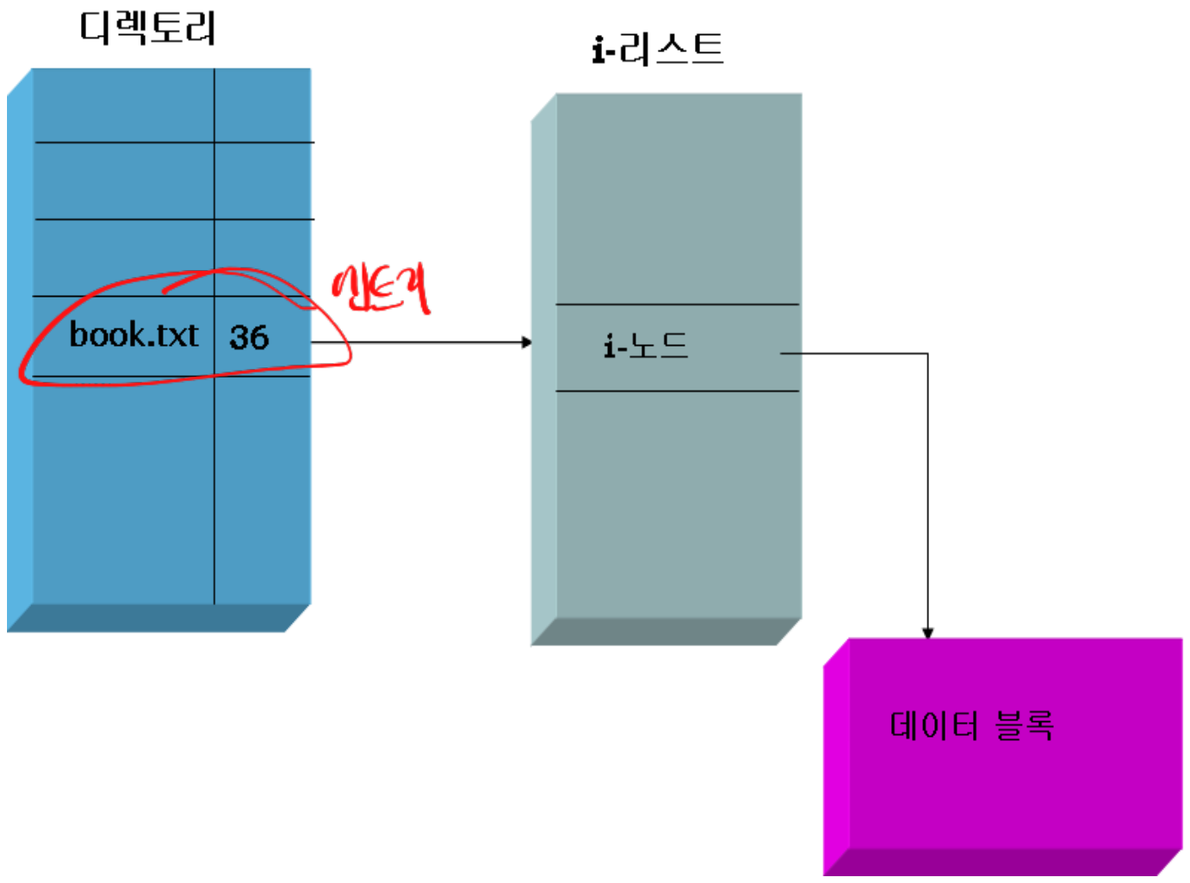
- 하나의 i-노드 내의 블록 포인터
  - 직접 블록 포인터 12개
  - 간접 블록 포인터 1개
  - 이중 간접 블록 포인터 1개
- 최대 몇 개의 데이터 블록을 가리킬 수 있을까?

12.3 디렉터리

디렉터리 구현

- 디렉터리 내에는 무엇이 저장되어 있을까? → 파일 이름, 해당 파일의 i-노드번호 (=엔트리)
- 디렉터리 엔트리

```
#include <dirent.h>
struct dirent {
  ino_t d_ino; // i-노드 번호
  char d_name[NAME_MAX + 1];
  // 파일 이름
}
```

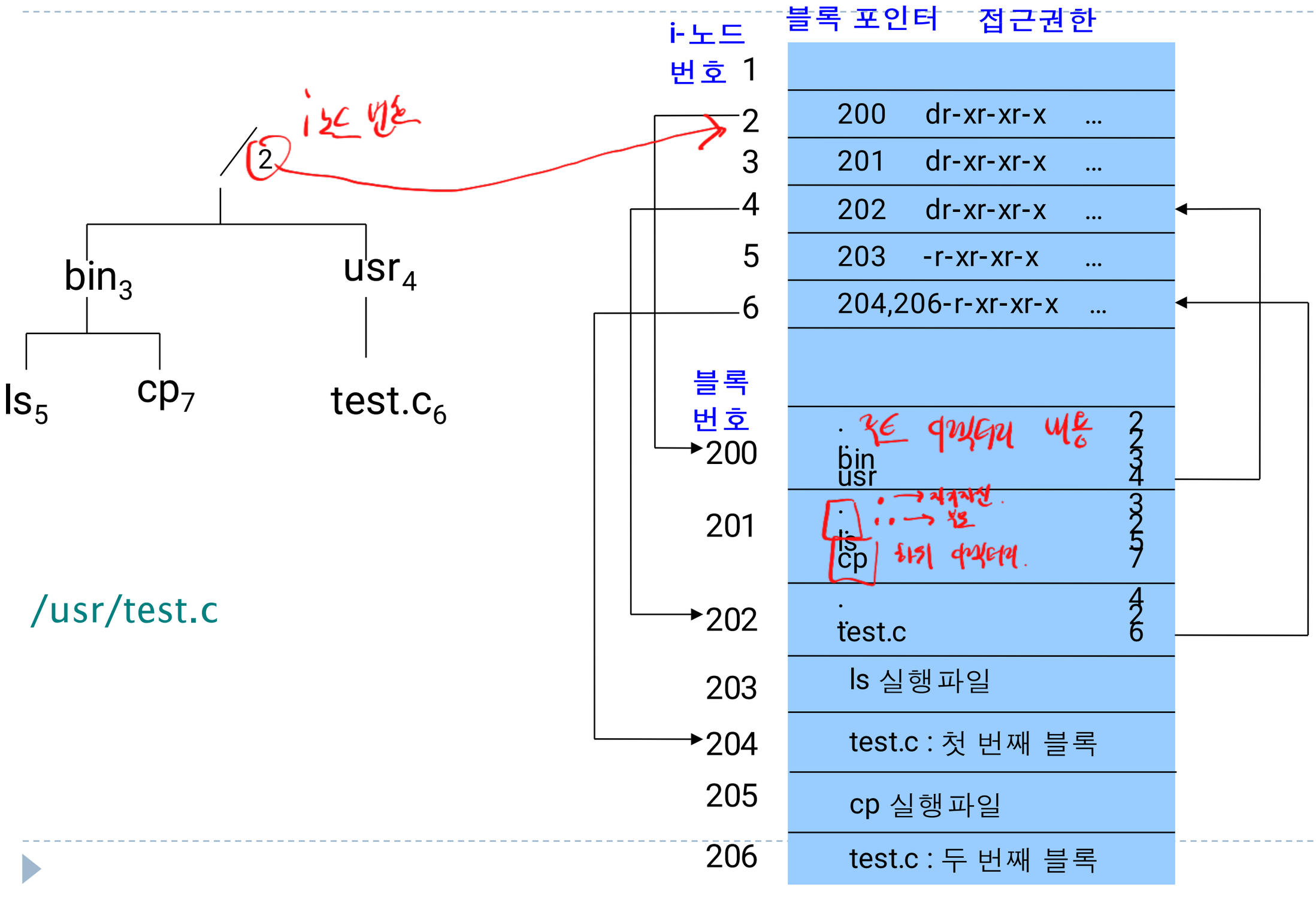


디렉터리 구현

- 그림의 파일 시스템 구조를 보자.
  - 디렉터리를 위한 별도의 구조는 없다.

- 파일 시스템 내에서 디렉터리를 어떻게 구현할 수 있을까?
  - 디렉터리도 일종의 파일로 다른 파일처럼 구현된다.
  - 디렉터리도 다른 파일처럼 하나의 i-노드로 표현된다.
  - 디렉터리의 내용은 디렉터리 엔트리(파일이름, i-노드 번호)

디렉토리 구현


i-노드 번호
블록 포인터
접근권한
2
bin3
usr4
ls5
cp7
test.c6
/usr/test.c
1
2
3
4
5
6
200
201
202
203
204
205
206
200 dr-xr-xr-x ...
201 dr-xr-xr-x ...
202 dr-xr-xr-x ...
203 -r-xr-xr-x ...
204,206-r-xr-xr-x ...
블록 번호
. 2
.. 2
bin 3
usr 4
. 3
.. 2
ls 5
cp 7
. 4
.. 2
test.c 6
ls 실행파일
test.c : 첫 번째 블록
cp 실행파일
test.c : 두 번째 블록

12.4 링크의 구현

링크

- 링크
  - 기존 파일에 대한 또 하나의 새로운 이름
  - 하드 링크 심볼릭 링크

- 사용법

$ ln [-s] 파일1 파일2

파일1에 대한 새로운 이름(링크)로 파일2를 만들어 준다. -s 옵션은 심볼릭 링크

$ ln [-s] 파일1 디렉터리

파일1에 대한 링크를 지정된 디렉터리에 같은 이름으로 만들어 준다.

하드 링크(hard link)

- 하드 링크
  - 기존 파일에 대한 새로운 이름이라고 생각할 수 있다.
  - 실제로 기존 파일을 대표하는 i-노드를 가리켜 구현한다.

- 예

  $ ln hello.txt hi.txt

  $ ls -l

  -rw------- 2 chang cs 15 11월 7일 15:31 hello.txt

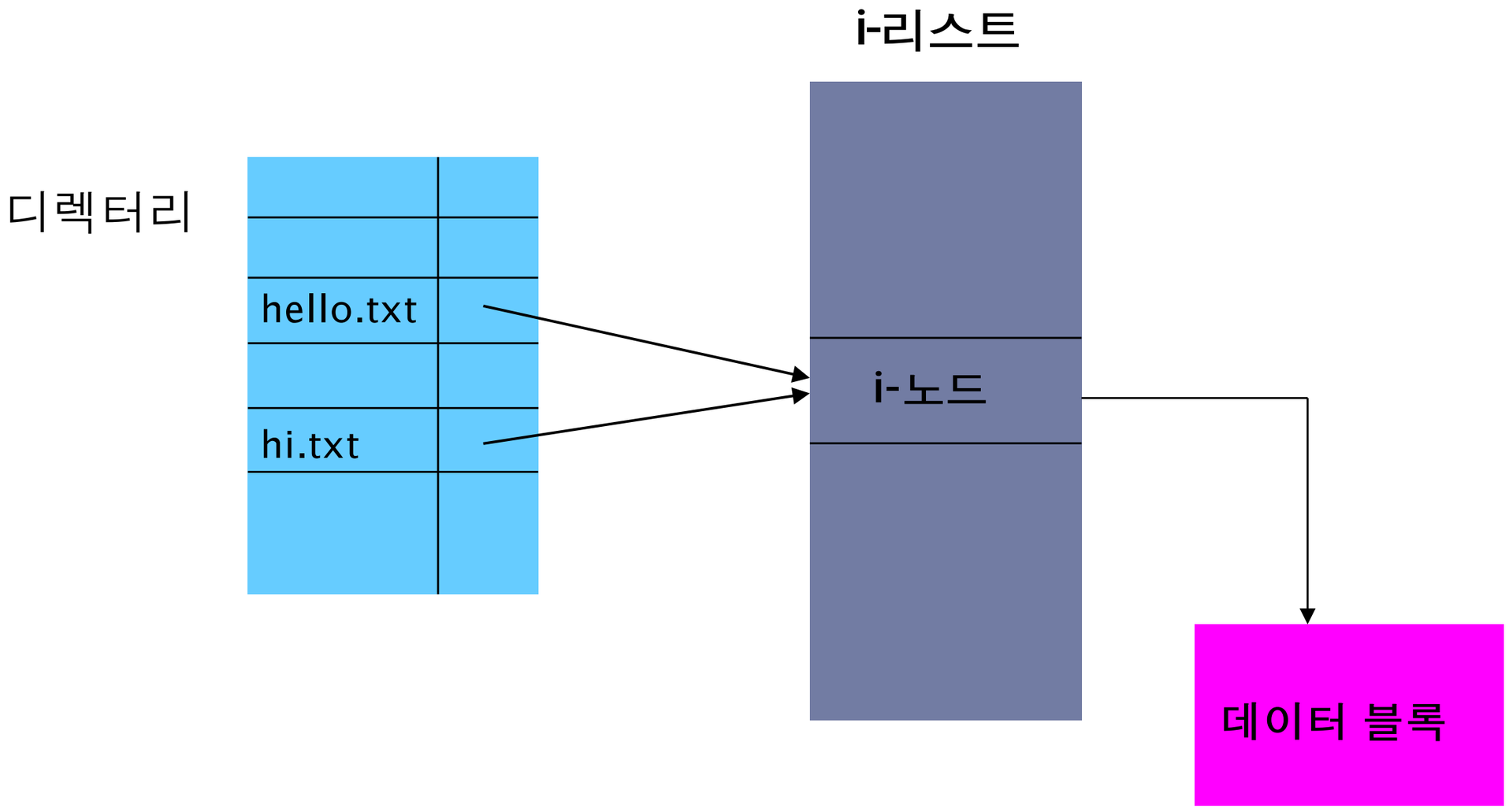
  -rw------- 2 chang cs 15 11월 7일 15:31 hi.txt

- 확인

  $ ls -i hello.txt hi.txt

  537384090 hello.txt 537384090 hi.txt

하드 링크 구현



심볼릭 링크(symbolic link)

- 심볼릭 링크
  - 다른 파일을 가리키고 있는 별도의 파일이다.
  - 실제 파일의 경로명을 저장하고 있는 일종의 특수 파일이다.
  - 이 경로명이 다른 파일에 대한 간접적인 포인터 역할을 한다.
- 예

```
$ ln -s hello.txt hi.txt
$ ls -l
-rw------- 1 chang chang 15 11월 7일 15:31 hello.txt
lrwxrwxrwx 1 chang chang 9 1월 24일 12:56 hi.txt -> hello.txt
```

심볼릭 링크: 예

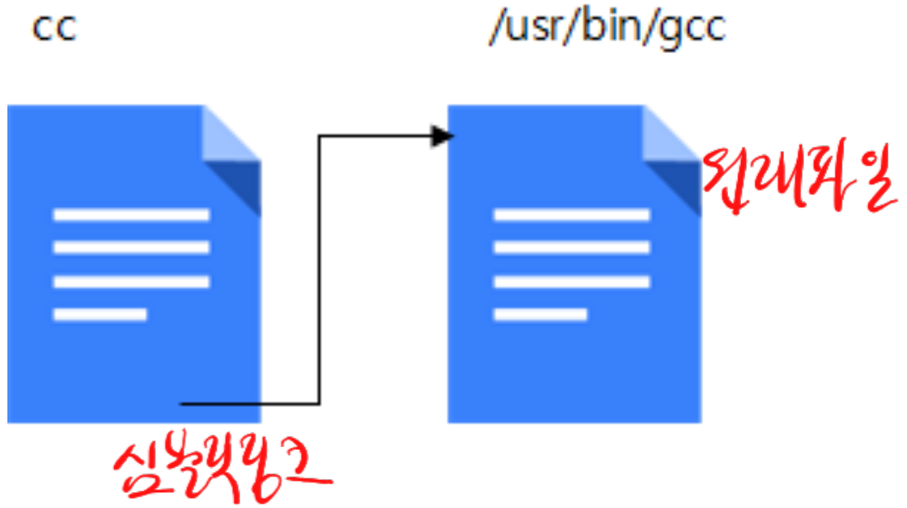
원본파일

$ ln –s /usr/bin/gcc cc

가르키는 파일의 별도의 파일.

$ ls –l cc

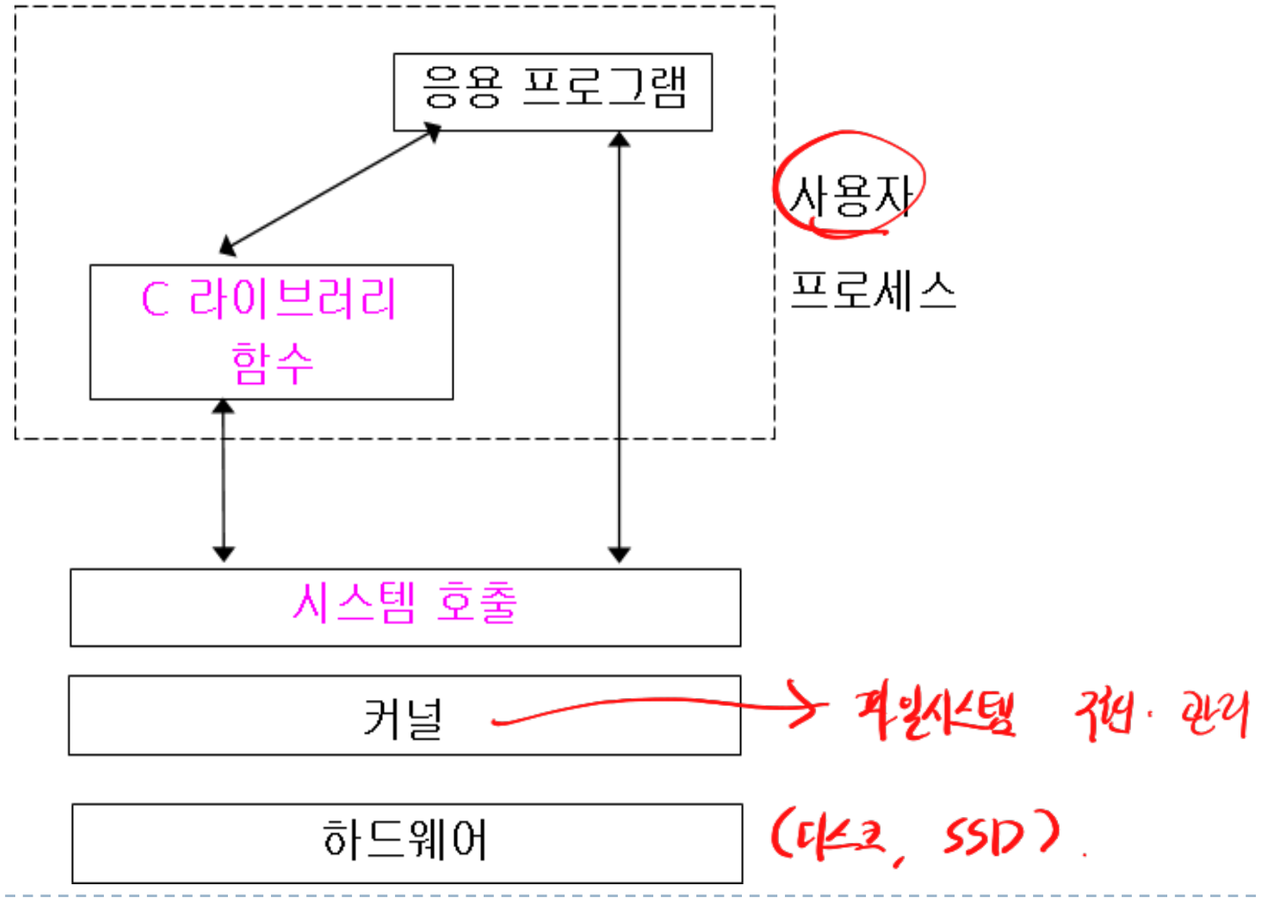
lrwxrwxrwx. 1 chang chang 12 7월 21 20:09 cc -> /usr/bin/gcc



12.5 C 파일 입출력

시스템 호출(system call)

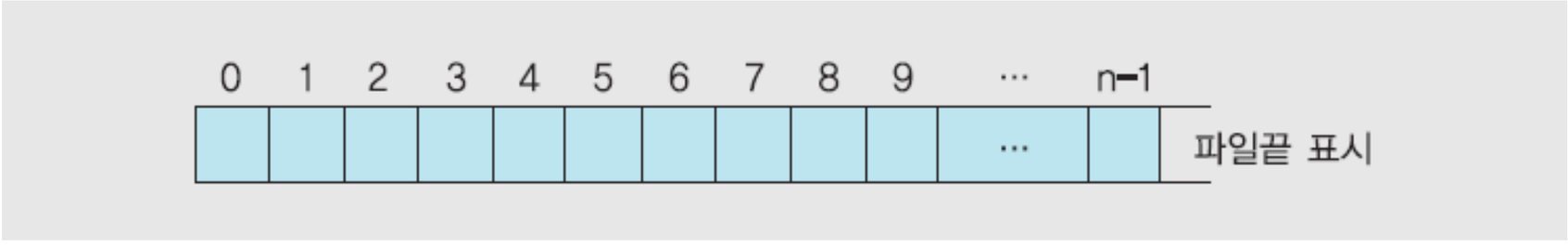
- 시스템 호출은 유닉스 커널에 서비스를 요청하기 위한 프로그래밍 인터페이스
- 응용 프로그램은 시스템 호출을 통해서 유닉스 커널에 서비스를 요청한다.



파일

- C 프로그램에서 파일은 왜 필요할까?
  - 변수에 저장된 정보들은 실행이 끝나면 모두 사라진다.
  - 정보를 영속적으로 저장하기 위해서는 파일에 저장해야 한다.

- 유닉스 파일
  - 모든 데이터를 연속된 바이트 형태로 저장한다.



C 언어의 파일 종류

- 텍스트 파일(text file)
  - 사람들이 읽을 수 있는 문자들을 저장하고 있는 파일
  - 텍스트 파일에서 "한 줄의 끝"을 나타내는 표현은 파일이 읽어 들여질 때, C 내부의 방식으로 변환된다.

- 이진 파일(binary file)
  - 모든 데이터는 있는 그대로 바이트의 연속으로 저장
  - 이진 파일을 이용하여 메모리에 저장된 변수 값 형태 그대로 파일에 저장할 수 있다.

파일 입출력

- C 언어의 파일 입출력 과정
  1. 파일 열기
     fopen( ) 함수 사용
  2. 파일 입출력
     다양한 파일 입출력 함수 사용
  3. 파일 닫기
     fclose( ) 함수 사용

파일 열기

- 파일을 사용하기 위해서는
  - 반드시 먼저 파일 열기(fopen)를 해야 한다.
  - 파일 열기를 하면 FILE 구조체에 대한 포인터가 리턴된다.
  - FILE 포인터는 열린 파일을 지정한다.

- 함수 fopen()

모드 (읽기 · 쓰기 · ·)

```
FILE *fopen(const char *filename, const char *mode);
```

파일 열기를 한다. 열린 파일을 나타내는 FILE 포인터를 반환한다. 오류가 발생하면 NULL을 반환한다.

파일 열기

- 예

```
FILE *fp;
fp = fopen(“~/work/text.txt", "r");
if (fp == NULL) {
    printf("파일 열기 오류\n");
}
```

- 예
  - fp = fopen("outdata.txt", "w");
  - fp = fopen("outdata.txt", "a");

fopen (): 텍스트 파일 열기

모드	의미	파일이 없으면	파일이 있으면
"r"	읽기 전용(read)	NULL 반환	정상 동작
"w"	쓰기 전용(write)	새로 생성	기존 내용 삭제
"a"	추가 쓰기(append)	새로 생성	기존 내용 뒤에 추가
"r+"	읽기와 쓰기	NULL 반환	정상 동작
"w+"	읽기와 쓰기	새로 생성	기존 내용 삭제
"a+"	추가를 위한 읽기와 쓰기	새로 생성	기존 내용 뒤에 추가

스트림과 FILE 구조체

- 스트림
  - 파일이 열리면 스트림(stream)이라고 한다.
- FILE 구조체
  - stdio.h에 정의되어 있음.
  - 열린 파일의 현재 상태를 나타내는 필드 변수들
  - 특히 파일 입출력에 사용되는 버퍼 관련 변수들

```
typedef struct {
   int cnt;                  // 버퍼의 남은 문자 수
   unsigned char*base;       // 버퍼 시작
   unsigned char*ptr;        // 버퍼의 현재 포인터
   unsigned flag;            // 파일 접근 모드
   int fd;                           // 열린 파일 디스크립터
} FILE;// FILE 구조체
```

buffer

열린 파일을 가리킨다 (파일 번호)

표준 입출력 스트림

- stdin, stdout, stderr
  - 각각 표준 입력, 표준 출력, 표준 오류를 나타내는 FILE 포인터
  - C 프로그램이 실행되면 자동적으로 열리고 프로그램이 종료될 때 자동으로 닫힘.

표준 입출력 스트림	설명	가리키는 장치
stdin	표준입력에 대한 FILE 포인터	키보드
stdout	표준출력에 대한 FILE 포인터	모니터
stderr	표준 오류에 대한 FILE 포인터	모니터

파일 닫기

- 파일을 열어서 사용한 후에는 파일을 닫아야 한다.

```
int fclose(FILE *fp );
```

fp가 가리키는 파일을 닫는다.

성공하면 0, 오류일 때는 -1을 반환한다.

- 예
  - fclose(fp);

12.6 파일 입출력 함수

파일 입출력 함수

표준 입출력 함수	파일 입출력 함수	기능
getchar()	fgetc(), getc()	문자단위로 입력하는 함수
putchar()	fputc(), putc()	문자단위로 출력하는 함수
gets()	fgets()	문자열을 입력하는 함수
puts()	fputs()	문자열을 출력하는 함수
scanf()	fscanf()	자료형에 따라 자료를 입력하는 함수
printf()	fprintf()	자료형에 따라 자료를 출력하는 함수

문자 단위 입출력

- fgetc() 함수와 fputc() 함수
  - 파일에 문자 단위 입출력을 할 수 있다.

- int fgetc(FILE *fp);
  - getc 함수는 fp가 지정한 파일에서 한 문자를 읽어서 리턴한다.
  - 파일 끝에 도달했을 경우에는 EOF(-1)를 리턴한다.

- int fputc(int c, FILE *fp);
  - putc 함수는 파일에 한 문자씩 출력한다.
  - 리턴값으로 출력하는 문자 리턴
  - 출력시 오류가 발생하면 EOF(-1) 리턴

12.7 명령어 구현

cat.c

```
#include <stdio.h>
/* 텍스트 파일 내용을 표준출력에 프린트 */
int main(int argc, char *argv[])
{
  FILE *fp;
  int c;
  if (argc < 2)                    // 명령줄 인수가 없으면 표준입력 사용
    fp = stdin;
  else  fp = fopen(argv[1],"r");   // 읽기 전용으로 파일 열기
  c = getc(fp);                    // 파일로부터 문자 읽기
  while (c != EOF) {               // 파일끝이 아니면
    putc(c, stdout);               // 읽은 문자를 표준출력에 출력
    c = getc(fp);                  // 파일로부터 문자 읽기
  }
  fclose(fp);
  return 0;
}
```

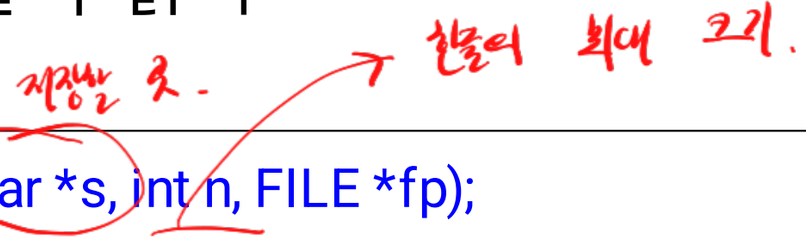
copy.c

```
#include <stdio.h>
/* 파일 복사 프로그램 */
int main(int argc, char *argv[])
{
  char c;
  FILE *fp1, *fp2;
  if (argc !=3) {
    fprintf(stderr, "사용법:
    %s 파일1 파일2\n", argv[0]);
    return 1;
  }
  fp1 = fopen(argv[1], "r");
  if (fp1 == NULL) {
    fprintf(stderr, "파일 %s 열기
    오류\n", argv[1]);
    return 2;
  }
  fp2 = fopen(argv[2], "w");
  while ((c = fgetc(fp1)) != EOF)
    fputc(c, fp2);
  fclose(fp1);
  fclose(fp2);
  return 0;
}
```

12.8 줄단위 입출력

줄 단위 입출력

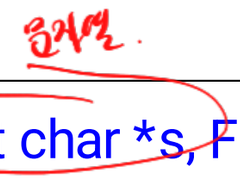
- 파일로 한 줄씩 읽기




```
char* fgets(char *s, int n, FILE *fp);
```

파일로부터 한 줄을 읽어서 문자열 포인터 s에 저장하고 s를 리턴한다.

- 파일에 한 줄씩 쓰기




```
int fputs(const char *s, FILE *fp);
```

문자열 s를 fp가 나타내는 파일에 출력한다. 성공하면 출력한 바이트 수를, 실패하면 EOF 값을 리턴한다.

line.c

```
#include <stdio.h>
#define MAXLINE 80
int main(int argc, char *argv[]) // 텍스트 파일에 줄 번호 붙여 프린트
{
    FILE *fp;
    int line = 0;
    char buffer[MAXLINE];
    ...
    if ( (fp = fopen(argv[1],"r")) == NULL) {
        fprintf(stderr, "파일 열기 오류\n");
        exit(2);
    }
    while (fgets(buffer, MAXLINE, fp) != NULL) {  // 한 줄 읽기
        line++;
        printf("%3d %s", line, buffer);    // 줄번호와 함께 프린트
    }
    exit(0);
}
```

핵심 개념

- 표준 유닉스 파일 시스템은 부트 블록, 슈퍼 블록, i-리스트, 데이터 블록 부분으로 구성된다.
- 파일 하나당 하나의 i-노드가 있으며 i-노드 내에 파일에 대한 모든 상태 정보가 저장되어 있다.
- 디렉터리는 일련의 디렉터리 엔트리들을 포함하고 각 디렉터리 엔트리는 파일 이름과 그 파일의 i-노드 번호로 구성된다.
- 링크는 기존 파일에 대한 또 다른 이름으로 하드 링크와 심볼릭(소프트) 링크가 있다.
- 시스템 호출은 유닉스 커널에 서비스를 요청하기 위한 프로그래밍 인터페이스로 응용 프로그램은 시스템 호출을 통해서 유닉스 커널에 서비스를 요청한다.