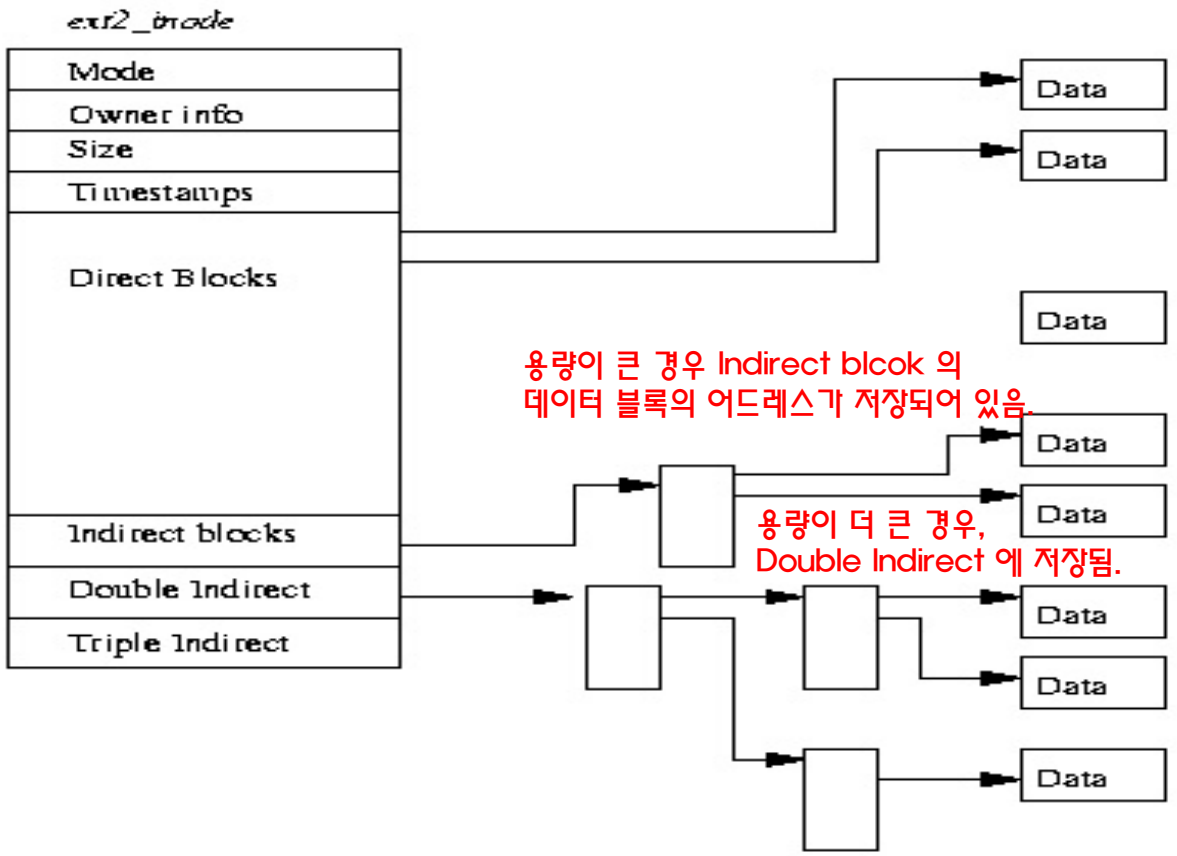
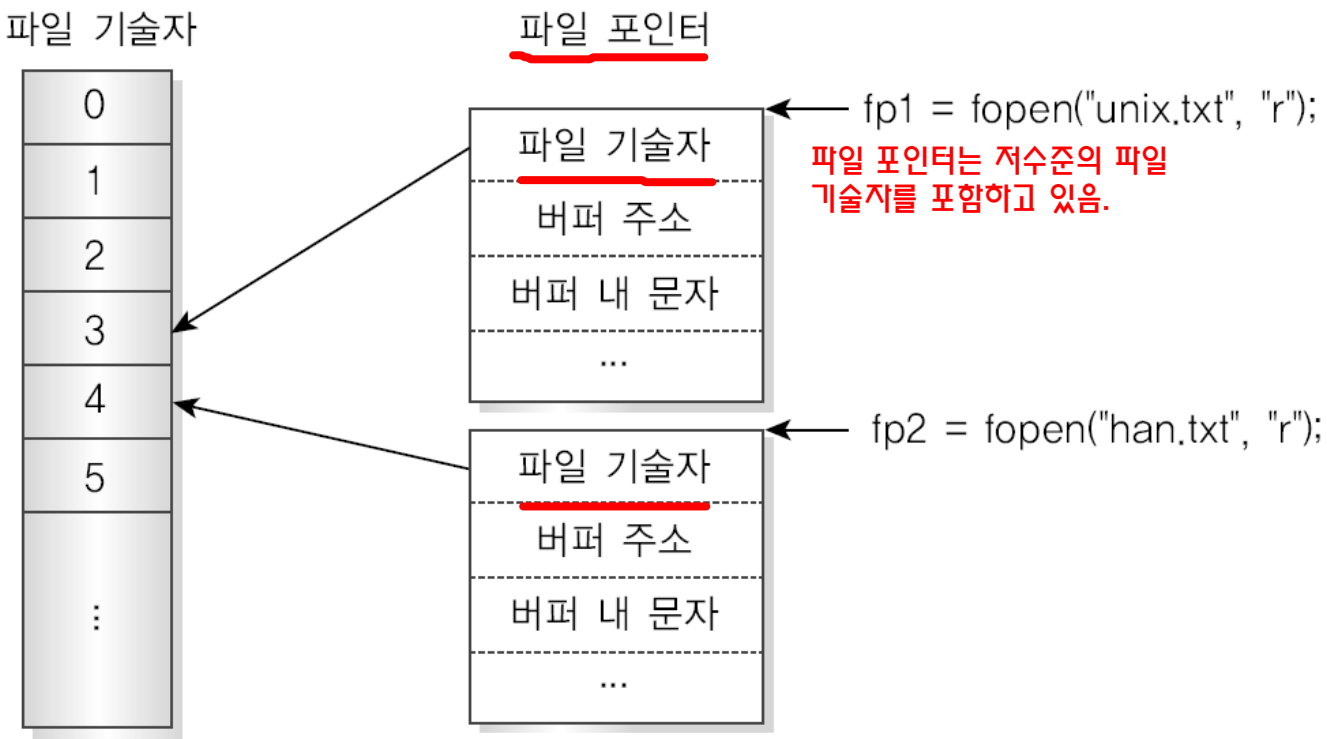
Inode 구조



모든 Inode 는 Inode number 를 가지고 있다.

파일 포인터

- 고수준 파일 입출력 : 표준 입출력 라이브러리
- 파일 포인터
  - 고수준 파일 입출력에서 열린 파일을 가리키는 포인터
  - 자료형으로 FILE * 형을 사용 -> 구조체에 대한 포인터



[그림 2-2] 파일 기술자와 파일 포인터

파일 열기와 닫기[1]

파일 열기: fopen(3)

```
#include <stdio.h>
FILE *fopen(const char *filename, const char *mode);
```

함수는 제시.

- filename으로 지정한 파일을 mode로 지정한 모드에 따라 열고 파일 포인터를 리턴
- mode 값

외울 필요는 없음. 단, 커맨드는 외워야 함.

ex) F_GETFL, F_SETFL

모드	의미
r	읽기 전용으로 텍스트 파일을 연다.
w	새로 쓰기용으로 텍스트 파일을 연다. 기존 내용은 삭제된다.
a	추가용으로 텍스트 파일을 연다.
rb	읽기 전용으로 바이너리 파일을 연다.
wb	새로 쓰기용으로 바이너리 파일을 연다. 기존 내용은 삭제된다.
ab	추가용으로 바이너리 파일을 연다.
r+	읽기와 쓰기용으로 텍스트 파일을 연다.
w+	쓰기와 읽기용으로 텍스트 파일을 연다.
a+	추가와 읽기용으로 텍스트 파일을 연다.
rb+	읽기와 쓰기용으로 바이너리 파일을 연다.
wb+	쓰기와 읽기용으로 바이너리 파일을 연다.
ab+	추가와 읽기용으로 바이너리 파일을 연다.

```
FILE *fp;
fp = fopen("unix.txt", "r");
```

파일 열기와 닫기[2]

- 파일 닫기: fclose(3)

```
#include  <stdio.h>
int  fclose(FILE  *stream);
```

  - fopen으로 오픈한 파일을 닫는다.

```
FILE  *fp;
fp  =  fopen("unix.txt",  "r");
fclose(fp);
```

문자 기반 입출력 함수

- 문자 기반 입력함수: fgetc(3), getc(3), getchar(3), getw(3)

참고로만 알아둬라.
문제로 안나온다.

```
#include <stdio.h>
int fgetc(FILE *stream);
int getc(FILE *stream);
int getchar(void);
int getw(FILE *stream);
```

  - fgetc : 문자 한 개를 unsigned char 형태로 읽어온다.
  - getc, getchar : 매크로
  - getw : 워드 단위로 읽어온다.

외울 필요는 없음.

- 문자 기반 출력함수: fputc(3), putc(3), putchar(3), putw(3)

```
#include <stdio.h>
int fputc(int c, *stream);
int putc(int c, *stream);
int putchar(int c);
int putw(int w, FILE *stream);
```

[예제 2-11] 문자 기반 입출력 함수 사용하기

```
01  #include <stdlib.h>
02  #include <stdio.h>
03
04  int main(void) {
05        FILE *rfp, *wfp;   int rfp, wfp
06        int c;
07
08        if ((rfp = fopen("unix.txt", "r")) == NULL) {
09              perror("fopen: unix.txt");
10              exit(1);
11        }
12
13        if ((wfp = fopen("unix.out", "w")) == NULL) {
14              perror("fopen: unix.out");
15              exit(1);
16        }
17
18        while ((c = fgetc(rfp)) != EOF) {
19              fputc(c, wfp);
20        }
21
22        fclose(rfp);
23        fclose(wfp);
24
25        return 0;
26  }
```

EOF를 만날 때까지 한 문자씩 읽어서 파일로 출력

```
# cat unix.txt
Unix System Programming
# ex2_11.out
# cat unix.out
Unix System Programming
```

문자열 기반 입출력

- 문자열 기반 입력 함수: gets(3), fgets(3) 문제로 안나온다.

```
#include <stdio.h>
char *gets(char *s);
char *fgets(char *s, int n, FILE *stream);
```

  - gets : 표준 입력에서 문자열을 읽어들인다.
  - fgets : 파일(stream)에서 n보다 하나 적게 문자열을 읽어 s에 저장

- 문자열 기반 출력 함수: puts(3), fputs(3)

```
#include <stdio.h>
char *puts(const char *s);
char *fputs(const char *s, FILE *stream);
```

[예제 2-12] 문자열 기반 입출력 함수 사용하기

ex2_12.c

```
01  #include <stdlib.h>
02  #include <stdio.h>
03
04  int main(void) {
05      FILE *rfp, *wfp;
06      char buf[BUFSIZ];
07
08      if ((rfp = fopen("unix.txt", "r")) == NULL) {
09          perror("fopen: unix.txt");
10          exit(1);
11      }
12
13      if ((wfp = fopen("unix.out", "a")) == NULL) {
14          perror("fopen: unix.out");
15          exit(1);
16      }
17
18      while (fgets(buf, BUFSIZ, rfp) != NULL) {
19          fputs(buf, wfp);
20      }
21
22      fclose(rfp);
23      fclose(wfp);
24
25      return 0;
26  }
```

한 행씩 buf로 읽어서 파일로 출력

```
# ex2_12.out
# cat unix.out
Unix System Programming
Unix System Programming
```

버퍼 기반 입출력

버퍼 기반 입력함수: fread(3)

외울 필요는 없지만 어떻게 쓰는지는 알아야 함.

```
#include <stdio.h>
size_t fread(void *ptr, size_t size, size_t nitems, FILE *stream);
```

- 항목의 크기가 size인 데이터를 nitems에 지정한 개수만큼 읽어 ptr에 저장
- 성공하면 읽어온 항목 수를 리턴
- 읽을 항목이 없으면 0을 리턴

버퍼 기반 출력함수: fwrite(3)

```
#include <stdio.h>
size_t fwrite(const void *ptr, size_t size, size_t nitems, FILE *stream);
```

- 항목의 크기가 size인 데이터를 nitems에서 지정한 개수만큼 ptr에서 읽어서 stream으로 지정한 파일에 출력
- 성공하면 출력한 항목의 수를 리턴
- 오류가 발생하면 EOF를 리턴

[예제 2-13] fread 함수로 파일 읽기

```
01 #include <stdlib.h>
02 #include <stdio.h>
03
04 int main(void) {
05 FILE *rfp;
06 char buf[BUFSIZ];
07 int n;
08
09 if ((rfp = fopen("unix.txt", "r")) == NULL) {
10 perror("fopen: unix.txt");
11 exit(1);
12 }
13
14 while ((n=fread(buf, sizeof(char)*2, 3, rfp)) > 0) {
15 buf[6] = '\0';
16 printf("n=%d, buf=%s\n", n, buf);
17 }
18
19 fclose(rfp);
20
21 return 0;
22 }
```

2개의 char 씩 3개를 읽음.

```
# ex2_13.out
n=3, buf=Unix S
n=3, buf=ystem
n=3, buf=Progra
n=3, buf=mming
```

[예제 2-14] fwrite 함수로 파일 출력하기

```
01  #include <stdlib.h>
02  #include <stdio.h>
03
04  int main(void) {
05      FILE *rfp, *wfp;
06      char buf[BUFSIZ];
07      int n;
08
09      if ((rfp = fopen("unix.txt", "r")) == NULL) {
10          perror("fopen: unix.txt");
11          exit(1);
12      }
13
14      if ((wfp = fopen("unix.out", "a")) == NULL) {
15          perror("fopen: unix.out");
16          exit(1);
17      }
18
19      while ((n = fread(buf, sizeof(char)*2, 3, rfp)) > 0) {
20          fwrite(buf, sizeof(char)*2, n, wfp);
21      }
22
23      fclose(rfp);
24      fclose(wfp);
25
26      return 0;
27  }
```

항목크기가 char크기의 2배. 이것을 3개.
즉 2*3=6바이트씩 읽어서 출력

제로를 리턴할 때 까지 반복

```
# ex2_14.out
# cat unix.out
Unix System Programming
```

형식 기반 입출력

형식 기반 입력 함수: scanf(3), fscanf(3)

알아두는게 좋다.

```
#include <stdio.h>
int scanf(const char *restrict format, ...);
int fscanf(FILE *restrict stream, const char *restrict format, ..);
```

- fscanf도 scanf가 사용하는 형식 지정 방법을 그대로 사용한다.
- 성공하면 읽어온 항목의 개수를 리턴한다.

형식 기반 출력 함수: printf(3), fprintf(3)

```
#include <stdio.h>
int printf(const char *restrict format, /* args */ ...);
int fprintf(FILE *restrict stream, const char *restrict format, /*args */ ..)/
```

- fprintf는 지정한 파일로 형식 지정 방법을 사용하여 출력한다.

[예제 2-15] fscanf 함수 사용하기

```
01  #include  <stdlib.h>
02  #include  <stdio.h>
03
04  int  main(void)  {
05  FILE  *rfp;
06  int  id,  s1,  s2,  s3,  s4,  n;
07
08  if  ((rfp  =  fopen("unix.dat",  "r"))  ==  NULL)  {
09  perror("fopen:  unix.dat");
10  exit(1);
11  }
12
13  printf("학번 평균\n");
14  while  ((n=fscanf(rfp,  "%d  %d  %d  %d  %d",  &id,&s1,&s2,&s3,&s4))  !=
    EOF)  {
15  printf("%d  :  %d\n",  id,  (s1+s2+s3+s4)/4);
16  }
17
18  fclose(rfp);
19
20  return  0;
21  }
```

```
#  cat  unix.dat
2009001  80  95  80  95
2009002  85  90  90  80
#  ex2_15.out
학번 평균
2009001  :  87
2009002  :  86
```

[예제 2-16] fprintf 함수 사용하기

```
01  #include <stdlib.h>
02  #include <stdio.h>
03
04  int main(void) {
05      FILE *rfp, *wfp;
06      int id, s1, s2, s3, s4, n;
07
08      if ((rfp = fopen("unix.dat", "r")) == NULL) {
09          perror("fopen: unix.dat");
10          exit(1);
11      }
12
13      if ((wfp = fopen("unix.scr", "w")) == NULL) {
14          perror("fopen: unix.scr");
15          exit(1);
16      }
17
18      fprintf(wfp, "   학번      평균\n");
19      while ((n=fscanf(rfp, "%d %d %d %d %d", &id,&s1,&s2,&s3,&s4)) != EOF) {
20          fprintf(wfp, "%d : %d\n", id, (s1+s2+s3+s4)/4);
21      }
22
23      fclose(rfp);
24      fclose(wfp);
25
26      return 0;
27  }
```

입출력에 형식 지정 기호 사용

```
# cat unix.dat
2009001 80 95 80 95
2009002 85 90 90 80
```

```
# ex2_16.out
# cat unix.scr
   학번     평균
2009001 : 87
2009002 : 86
```

파일 오프셋 지정[1]

파일 오프셋 이동: fseek(3)

알아둬야 함.

```
#include <stdio.h>
int fseek(FILE *stream, long offset, int whence);
```

SEEK_SET, SEEK_CUR, SEEK_END

- stream이 가리키는 파일에서 offset에 지정한 크기만큼 오프셋을 이동
- whence는 lseek와 같은 값을 사용
- fseek는 성공하면 0을 실패하면 EOF를 리턴

현재 오프셋 구하기: ftell(3)

```
#include <stdio.h>
long ftell(FILE *stream);
```

파일 오프셋 지정[2]

- 처음 위치로 오프셋 이동: rewind(3)

```
#include  <stdio.h>
void  rewind(FILE  *stream);
```

  - 오프셋을 파일의 시작 위치로 즉시 이동

- 오프셋의 저장과 이동: fsetpos(3), fgetpos(3)

알아두는게 좋다.

```
#include  <stdio.h>
int  fsetpos(FILE  *stream,  const  fpos_t  *pos);
int  fgetpos(FILE  *stream,  fpos_t  *pos);
```

  - fgetpos : 파일의 현재 오프셋을 pos가 가리키는 장소에 저장
  - fsetpos : pos가 가리키는 위치로 파일 오프셋을 이동

ls -al >> unix.txt
를 하면 append 됨.

[예제 2-17] fseek 함수 사용하기

```
...
04   int  main(void)  {
05          FILE  *fp;
06           int  n;
07          long  cur;
08          char  buf[BUFSIZ];
09
10           if  ((fp  =  fopen("unix.txt",  "r"))  ==  NULL)  {
11                 perror("fopen:  unix.txt");
12                 exit(1);
13          }
14
•           cur  =  ftell(fp);
•          printf("Offset  cur=%d\n",  (int)cur);
17
18           n  =  fread(buf,  sizeof(char),  4,  fp);
19           buf[n]  =  '\0';
20           printf("--  Read  Str=%s\n",  buf);
21
22          fseek(fp,  1,  SEEK_CUR);
23
24           cur  =  ftell(fp);
25           printf("Offset  cur=%d\n",  (int)cur);
26
```

현재 오프셋 읽기

오프셋 이동

[예제 2-17] fseek 함수 사용하기

```
27          n = fread(buf, sizeof(char), 6, fp);
28          buf[n] = '\0';
29          printf("-- Read Str=%s\n", buf);
30
31          cur = 12;
32          fsetpos(fp, &cur);
33
34          fgetpos(fp, &cur);
35          printf("Offset cur=%d\n", (int)cur);
36
37          n = fread(buf, sizeof(char), 13, fp);
38          buf[n] = '\0';
39          printf("-- Read Str=%s\n", buf);
40
41          fclose(fp);
42
43          return 0;
44      }
```

오프셋 이동

현재 오프셋 읽어서 지정

```
# ex2_17.out
Offset cur=0
-- Read Str=Unix
Offset cur=5
-- Read Str=System
Offset cur=12
-- Read Str=Programming
```

요약

파일

inode 와 같이 데이터 블럭이 저장됨.

- 파일은 관련 있는 데이터들의 집합으로 하드디스크 같은 저장장치에 일정한 형태로 저장된다.
- 유닉스에서 파일은 데이터를 저장하기 위해서 뿐만 아니라 데이터를 전송하거나 장치에 접근하기 위해서도 사용한다.

저수준 파일 입출력과 고수준 파일 입출력

- 저수준 파일 입출력 : 유닉스 커널의 시스템 호출을 사용하여 파일 입출력을 실행하며, 특수 파일도 읽고 쓸 수 있다.
- 고수준 파일 입출력 : 표준 입출력 라이브러리로 다양한 형태의 파일 입출력 함수를 제공한다.

알아두면 좋다.

	저수준 파일 입출력	고수준 파일 입출력
파일 지시자	int fd (파일 기술자)	FILE *fp; (파일 포인터)
특징	• 더 빠르다. • 바이트 단위로 읽고 쓴다. • 특수 파일에 대한 접근이 가능하다.	• 사용하기 쉽다. • 버퍼 단위로 읽고 쓴다. • 데이터의 입출력 동기화가 쉽다. • 여러 가지 형식을 지원한다.

유닉스 파일의 특징[1]

파일
- 유닉스에서 파일은 데이터 저장, 장치구동, 프로세스 간 통신 등에 사용

파일의 종류
- 일반파일, 디렉토리, 특수파일

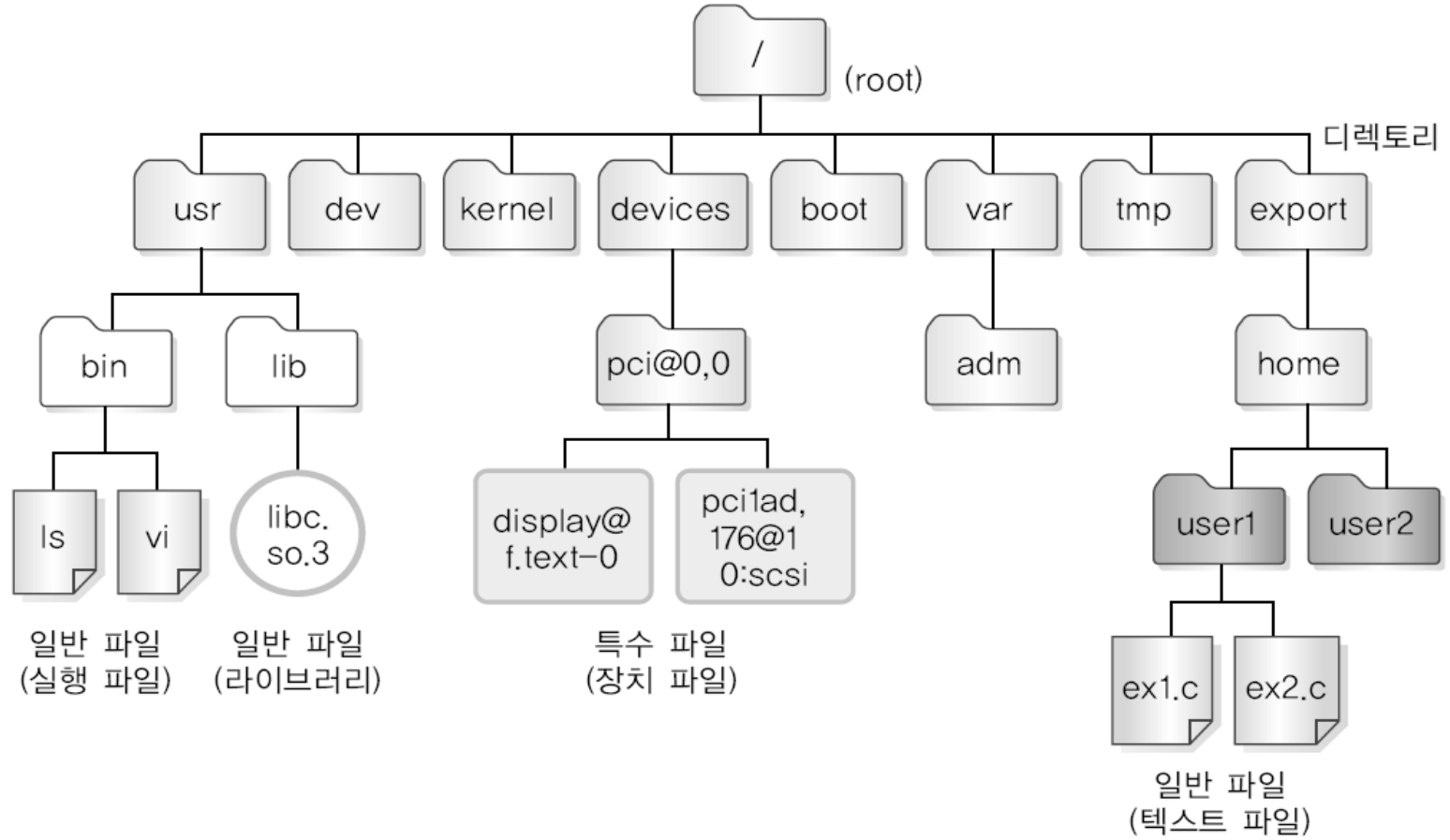
일반파일
- 텍스트 파일, 실행파일, 라이브러리, 이미지 등 유닉스에서 사용하는 대부분의 파일
- 편집기나 다른 응용 프로그램을 통해 생성

장치파일
- 장치를 사용하기 위한 특수 파일. 데이터 블록은 사용하지 않음
- 블록장치파일 : 블록단위(Soralis:8KB)로 읽고 쓴다.
- 문자장치파일 : 섹터단위(512바이트)로 읽고 쓴다 -> 로우디바이스(Raw Device)
- 예 : /devices

디렉토리
- 디렉토리도 파일로 취급
- 디렉토리와 관련된 데이터 블록은 해당 디렉토리에 속한 파일의 목록과 inode 저장

유닉스 파일의 특징[1]

파일 종류



[그림 3-1] 유닉스 파일의 종류

유닉스 파일의 특징[2]

파일의 종류 구분

- ls –l 명령으로 파일의 종류 확인 가능 : 결과의 맨 앞글자로 구분

```
# ls -l /usr/bin/vi
-r-xr-xr-x    5 root       bin          193968 2007   9월 14일 /usr/bin/vi
```

- 파일 종류 식별 문자

문자	파일의 종류
-	일반 파일
d	디렉토리
b	블록 장치 특수 파일
c	문자 장치 특수 파일
l	심볼릭 링크

- 예제

```
# ls -l /
lrwxrwxrwx    1 root root           9   7월 16일   15:22 bin -> ./usr/bin
drwxr-xr-x   42 root sys         1024   8월    5일    03:26 usr
......
# ls -lL /dev/dsk/c0d0s0
brw-r-----    1 root sys       102,   0   8월    3일   10:59 /dev/dsk/c0d0s0
# ls -lL /dev/rdsk/c0d0s0
crw-r-----    1 root sys       102,   0   8월    3일   12:12 /dev/rdsk/c0d0s0
```

유닉스 파일의 특징[3]

파일의 구성 요소

- 파일명, inode, 데이터블록

파일명

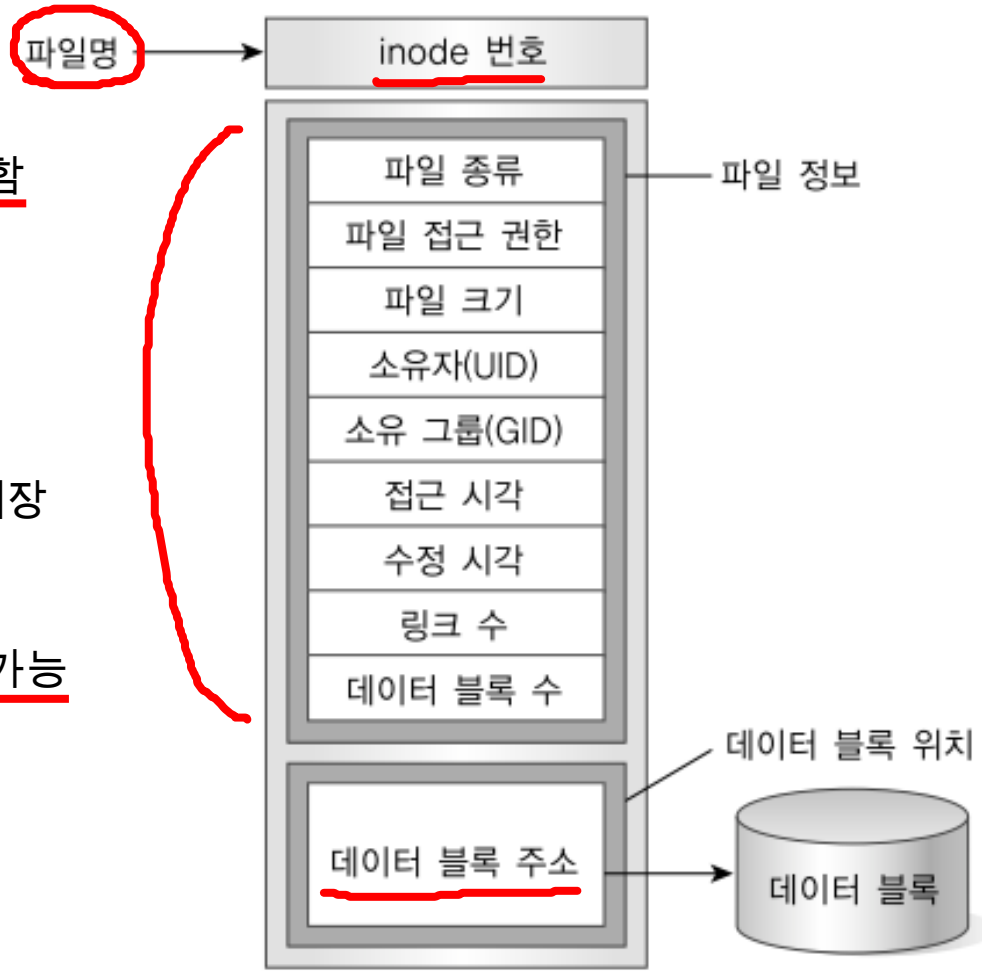
- 사용자가 파일에 접근할 때 사용
- 파일명과 관련된 inode가 반드시 있어야 함
- 파일명은 최대 255자까지 가능
- 파일명에서 대소문자를 구분

inode

- 외부적으로는 번호로 표시하며
  내부적으로는 두 부분으로 나누어 정보 저장
  - 파일 정보를 저장하는 부분
  - 데이터 블록의 주소 저장하는 부분
- 파일의 inode 번호는 ls –i 명령으로 확인 가능

데이터 블록

- 실제로 데이터가 저장되는 부분



파일 정보 검색[1]

- 파일명으로 파일 정보 검색 : stat(2)

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
int stat(const char *restrict path, struct stat *buf);
```

  - inode에 저장된 파일 정보 검색
  - path에 검색할 파일의 경로를 지정하고, 검색한 정보를 buf에 저장
  - stat 구조체
    제시해줄 것임.

```
struct stat {
        dev_tst_dev;
        ino_tst_ino;
        mode_tst_mode;
        nlink_tst_nlink;
        uid_tst_uid;
        gid_tst_gid;
        dev_tst_rdev;
        off_tst_size;
time_tst_atime;
        time_tst_mtime;
        time_tst_ctime;
        blksize_tst_blksize;
        blkcnt_tst_blocks;
        charst_fstype[_ST_FSTYPSZ];
};
```

[예제 3-1] 파일명으로 inode 정보 검색하기

```
01#include  <sys/types.h>
02#include  <sys/stat.h>
03#include  <stdio.h>
04
05int  main(void)  {
06struct  stat  buf;
07
08stat("unix.txt",  &buf);
09
10printf("Inode  =  %d\n",  (int)buf.st_ino);
11printf("Mode  =  %o\n",  (unsigned  int)buf.st_mode);
12printf("Nlink  =  %o\n",(unsigned  int)  buf.st_nlink);
13printf("UID  =  %d\n",  (int)buf.st_uid);
14printf("GID  =  %d\n",  (int)buf.st_gid);
15printf("SIZE  =  %d\n",  (int)buf.st_size);
16printf("Atime  =  %d\n",  (int)buf.st_atime);
17printf("Mtime  =  %d\n",  (int)buf.st_mtime);
18printf("Ctime  =  %d\n",  (int)buf.st_ctime);
19printf("Blksize  =  %d\n",  (int)buf.st_blksize);
20printf("Blocks  =  %d\n",  (int)buf.st_blocks);
21printf("FStype  =  %s\n",  buf.st_fstype);
22
23return  0;
24   }
```

정의해줘야 함.

```
#  ex3_1.out
Inode  =  192
Mode  =  100644
Nlink  =  1
UID  =  0
GID  =  1
SIZE  =  24
Atime  =  1231397228
Mtime  =  1231397228
Ctime  =  1231397228
Blksize  =  8192
Blocks  =  2
FStype  =  ufs
```

파일 정보 검색[2]

- 파일 기술자로 파일 정보 검색 : fstat(2)

```
#include  <sys/types.h>
#include  <sys/stat.h>
int  fstat(int  fd,  struct  stat  *buf);
```

파일 기술자

[예제 3-2] 명령행 인자 출력하기

```
01   #include  <sys/types.h>
02   #include  <sys/stat.h>
03   #include  <stdio.h>
04
05   int  main(void)  {
06         struct  stat  buf;
07
08         stat("unix.txt",  &buf);
09
10         printf("Inode  =  %d\n",  (int)buf.st_ino);
11  printf("Mode  =  %o\n",  (unsigned  int)buf.st_mode);
12  printf("Nlink  =  %o\n",(unsigned  int)  buf.st_nlink);
13         printf("UID  =  %d\n",  (int)buf.st_uid);
```

신경 안써도 된다.

명령행 인자가 필요한 경우에는 써줘야 한다.

반드시 써줘야 함.

```
fd = open("unix.txt", O_RDONLY);
fstat(fd, &buf);
```

[예제 3-2] fstat으로 파일 정보 검색하기

```
14          printf("GID  =  %d\n",  (int)buf.st_gid);
15          printf("SIZE  =  %d\n",  (int)buf.st_size);
16          printf("Atime  =  %d\n",  (int)buf.st_atime);
17          printf("Mtime  =  %d\n",  (int)buf.st_mtime);
18          printf("Ctime  =  %d\n",  (int)buf.st_ctime);
19          printf("Blksize  =  %d\n",  (int)buf.st_blksize);
20          printf("Blocks  =  %d\n",  (int)buf.st_blocks);
21          printf("FStype  =  %s\n",  buf.st_fstype);
22
23          return  0;
24      }
```

```
#  ex3_2.out
Inode  =  192
UID  =  0
```