제 5 장
디렉터리

ACS30021
고급 프로그래밍

나보균 (bkna@kpu.ac.kr)

컴퓨터 공학과
한국산업기술 대학교

학습 목표

- Directories
- The implementation of a directory
- Programming with directories

File System 연결(mount)

- 파티션의 장점
  - ✓ 신뢰성(Reliability) 향상
  - ✓ 서로 다른 종류의 파일 시스템 지원 가능 (NT와 Ext2)
  - ✓ 같은 파일 시스템일지라도 access pattern에 따라 다른 설정 가능
  - ✓ 특정 프로세스에 의한 디스크 공간 독점 방지

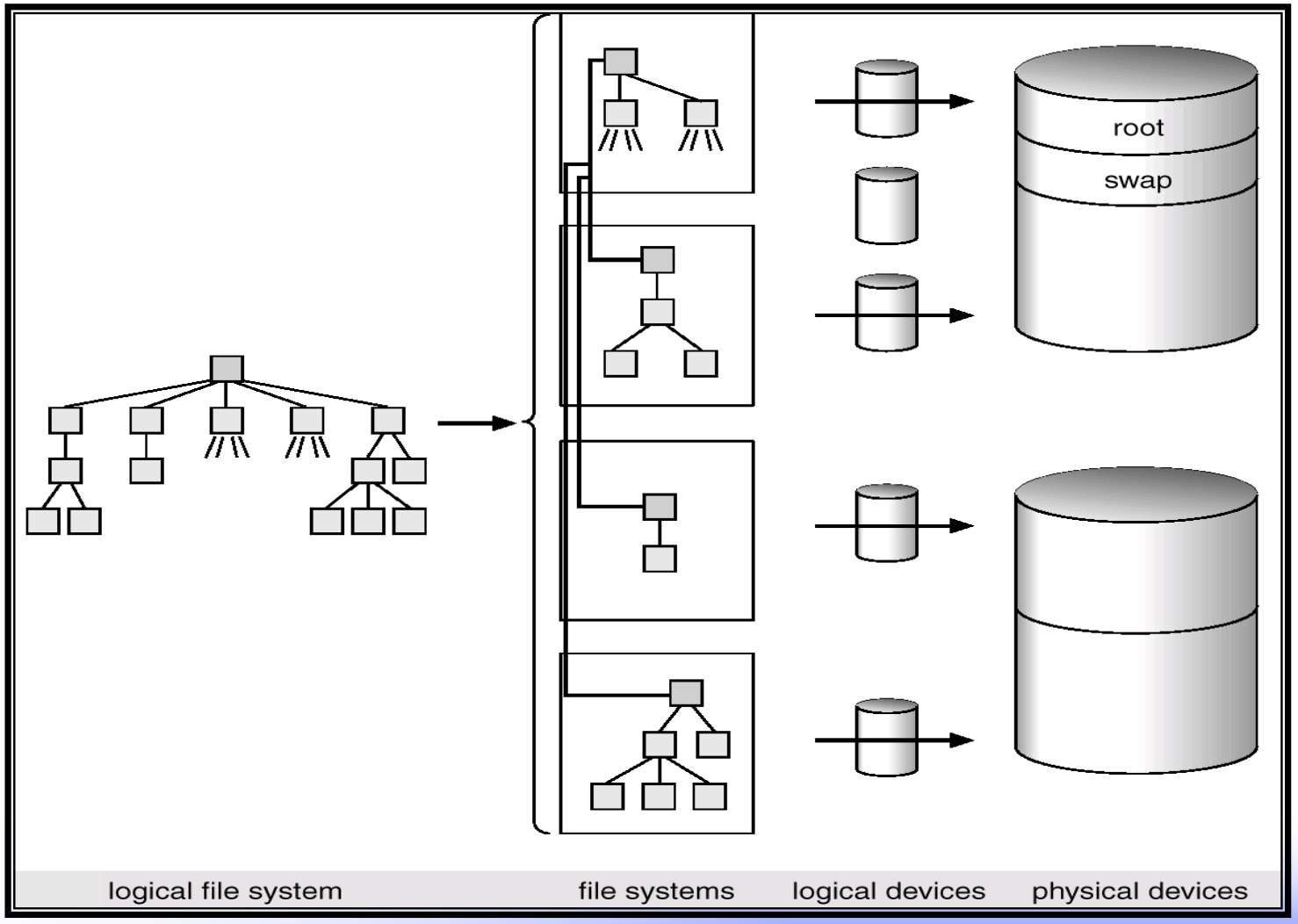
- 파티션을 지원하기 위해서는
  - ✓ 논리적인 하나의 파일 시스템이 실제로는 여러 디스크 or 파티션에 존재 가능

File System 연결(mount)

- File system 생성
  - ✓ format: /usr/bin/superformat
  - ✓ create: /sbin/mke2fs
    - ➢ initializes superblock and group descriptor
    - ➢ defective block on the partition 확인
    - ➢ 각 블록의 inode bit map과 data bit map 초기화
    - ➢ 각 블록의 inode table 초기화
    - ➢ root directory 생성
    - ➢ lost+found directory 생성

File System 연결(mount)

- 논리적인 파일 시스템



File System 연결(mount)

- 기준점 설정
  - ✓ root file system
  - ✓ 커널이 항상 접근 가능
  - ✓ 꼭 필요한 시스템 명령(eg: ifconfig)들은 root file system에 존재
- 파일 시스템 연결
  - ✓ mount: integrated into the directory hierarchy of the root file system
  - ✓ mounted point

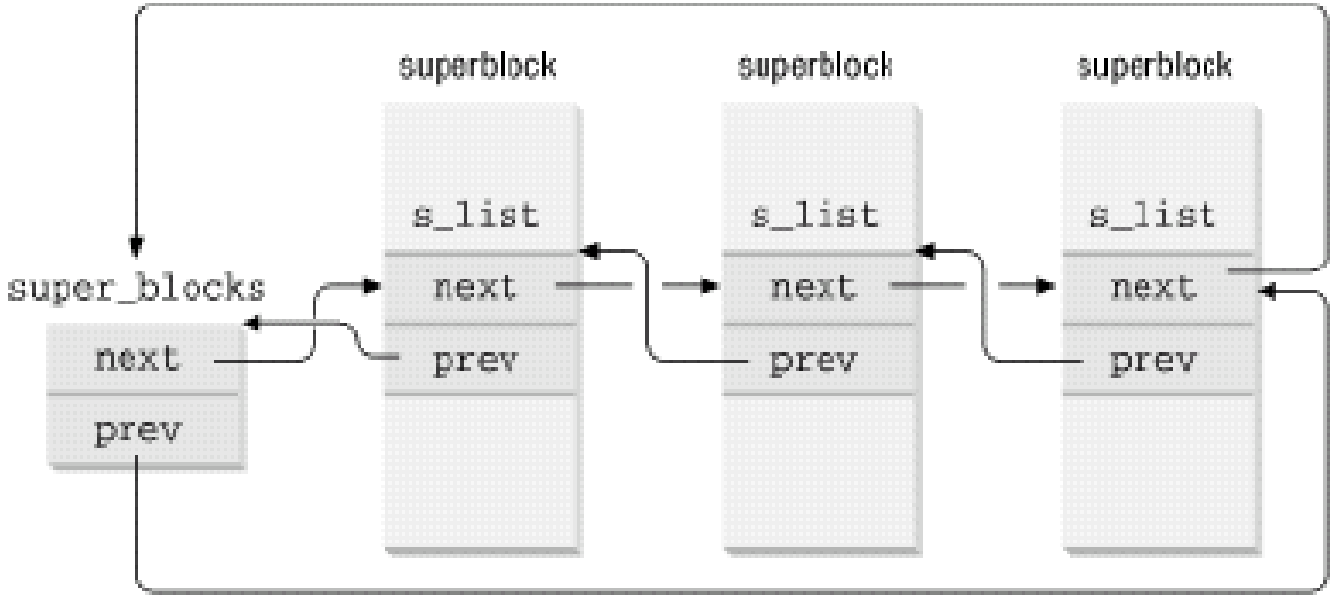
```
# /etc/fstab
# device      directory   type    options
#
/dev/hda1     /           ext2    defaults
/dev/hdb2     /home       ext2    defaults
/dev/hdb1     none        swap    sw
/proc         /proc       proc    defaults
```

File System 연결(mount)

- Virtual file system
  - mount 명령

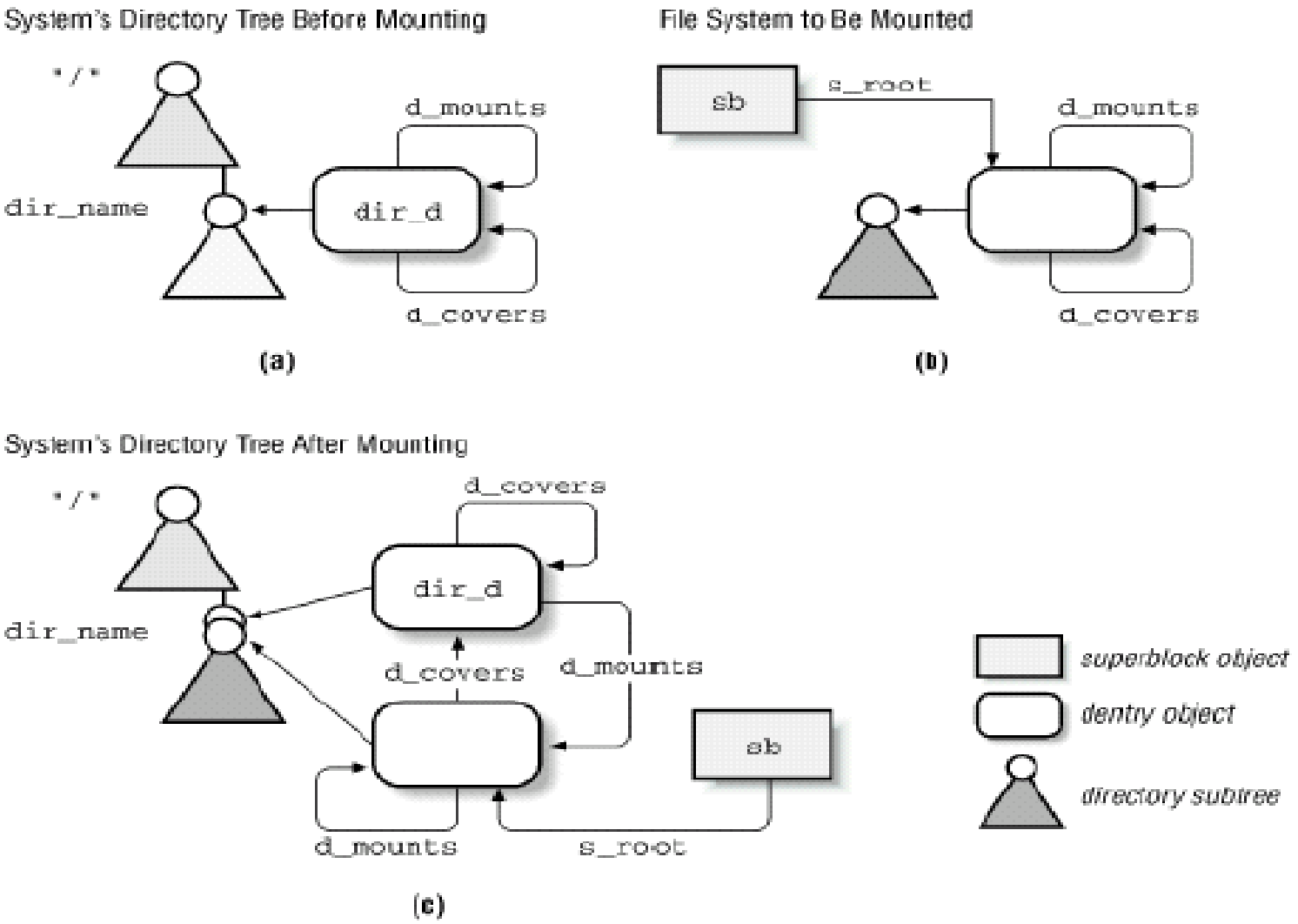
```
mount -t msdos /dev/fd0 /mnt
```

  - mount 결과



File System 연결(mount)

- mount 예

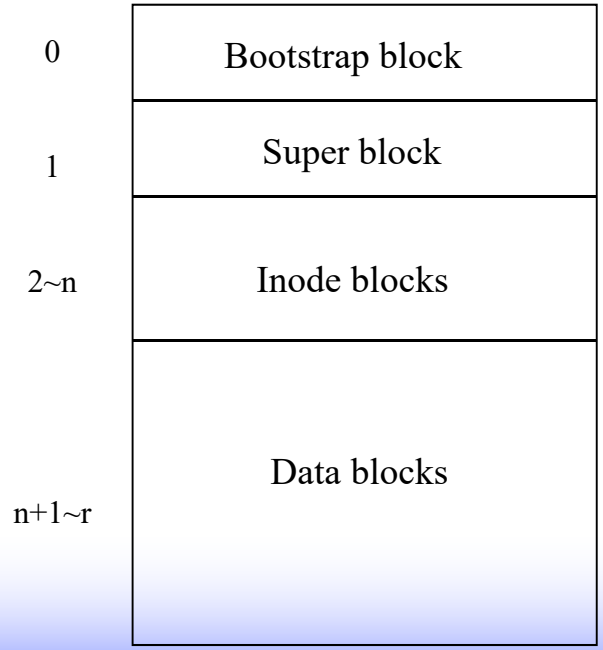


Directory, File system, Special file

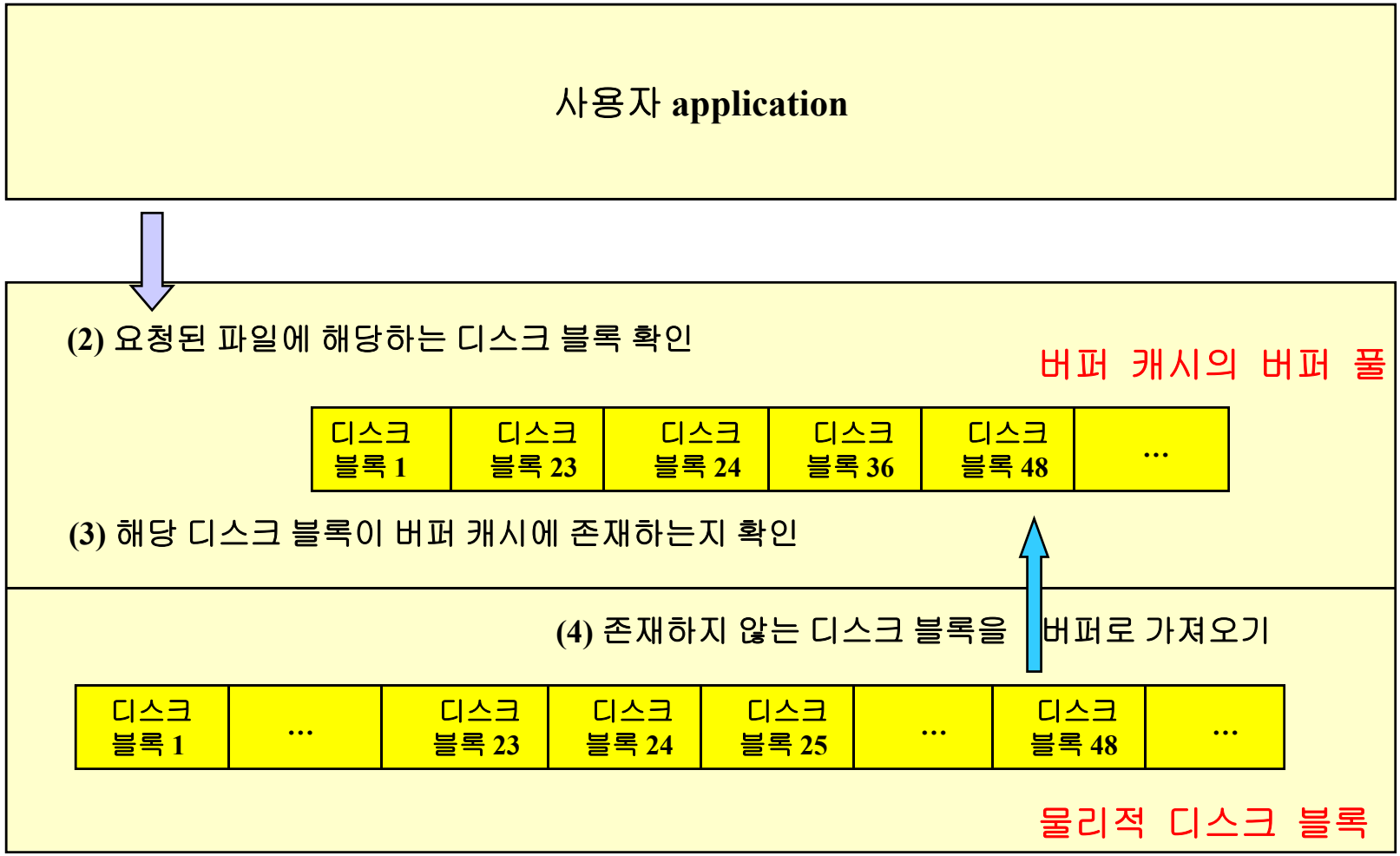
- *Directories* – Directories act as repositories for filename. Directories can be nested and this gives the file structure a hierarchical, tree-like form.
- *File system* – File systems are collections of files and directories. A file system corresponds to a physical section(partition) of a disk.
- *Special files* – UNIX extends the file concept to cover the **peripheral devices** connected to a system.
  - ✓ These peripheral devices such as printers, disk and even memory, are represented by filenames in the file structure.
  - ✓ These can be accessed using UNIX file I/O system calls.

UNIX File System

- File system layout
  - ✓ Bootstrap block(0)– kernel loading program
  - ✓ Super block(1) – file system information usually kept in memory
    - # of blocks for file system (r)
    - # of blocks for inode blocks (n–1)
    - # of blocks for data blocks (r–n)
    - File system last update time
    - A list of free data block numbers
    - A list of free inode numbers
  - ✓ Inode blocks(2 ~ n)
    - A list of inode structures
  - ✓ Data blocks(n+1 ~ r)
    - A list of data blocks



Caching: Buffer cache



UNIX Device Files

- Device files: each device has one file in `/dev`
  - 예: `/dev/lp0`, `/dev/console`, `/dev/hda1`
- Device files can be used as regular files
  - 예: `$ cat file > /dev/lp0`
- Block and character devices
  - ✓ Block(b) –disk, tape
  - 예: `brw-rw---- 1 root disk  3,1 May 6 1998 /dev/hda1`
  - ✓ Character(c) – terminal, modem, printer
  - 예: `crw-rw---- 1 root daemon 6,0 May 6 1998 /dev/lp0`
- Major and minor device numbers
  - ✓ Major – device driver number
  - ✓ Minor – device port number

UNIX Device Files: 예

```
#include<fcntl.h>

main()
{
 int i, fd;

 fd = open("/dev/tty00", O_WRONLY);

 for(i = 0; i< 100; i++)
       write(fd, "x", 1);

 close(fd);
}
```

Directory – User's View

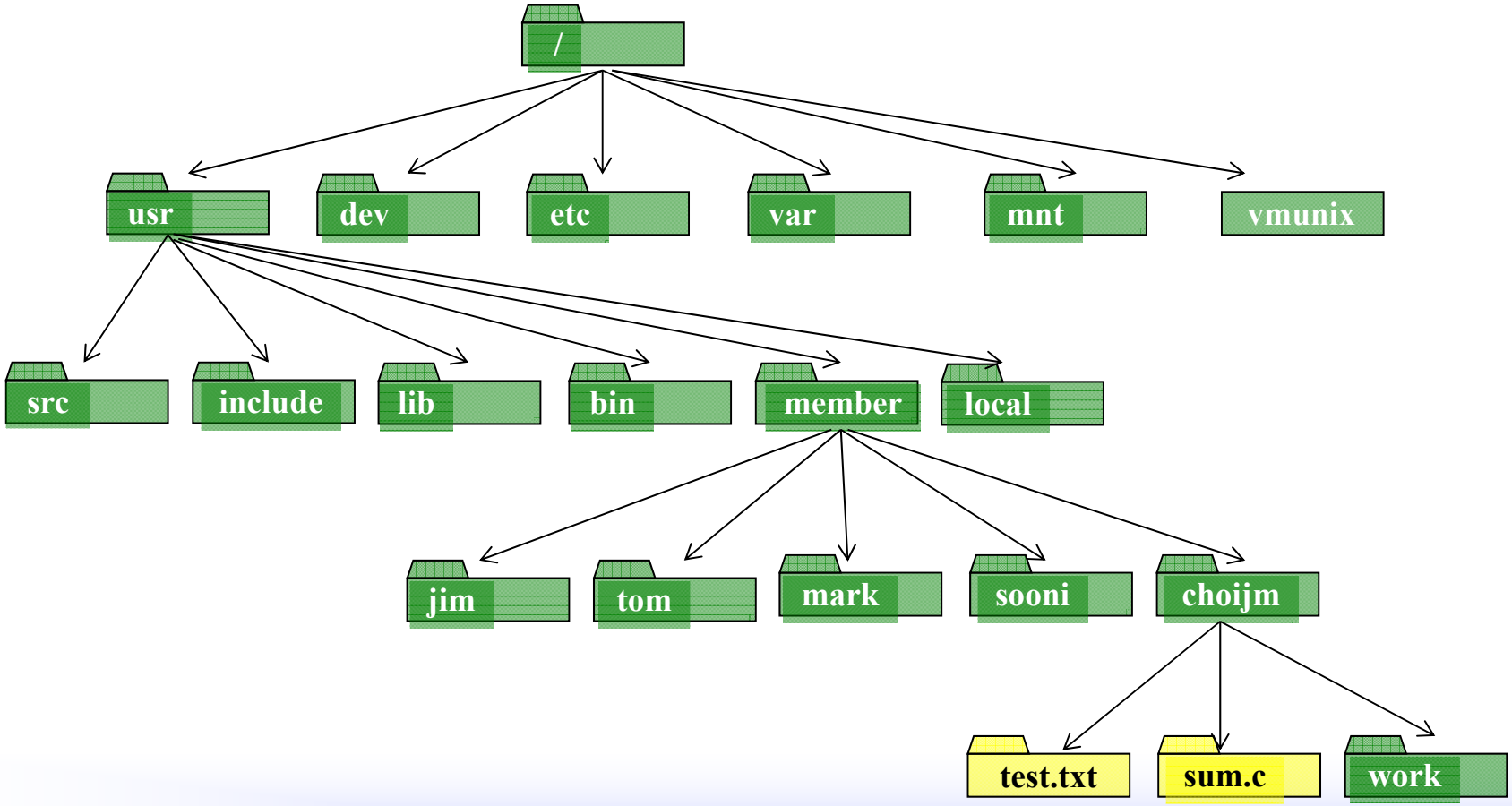
- *Home directory* – The directory where a user is placed at login.
- *Current working directory* – The directory where a user is currently working at.
- File's pathname
  - ✓ Absolute – from the root directory (eg: /home/john/book)
  - ✓ Relative – from the currently working directory (eg: ../john/book)

Directory

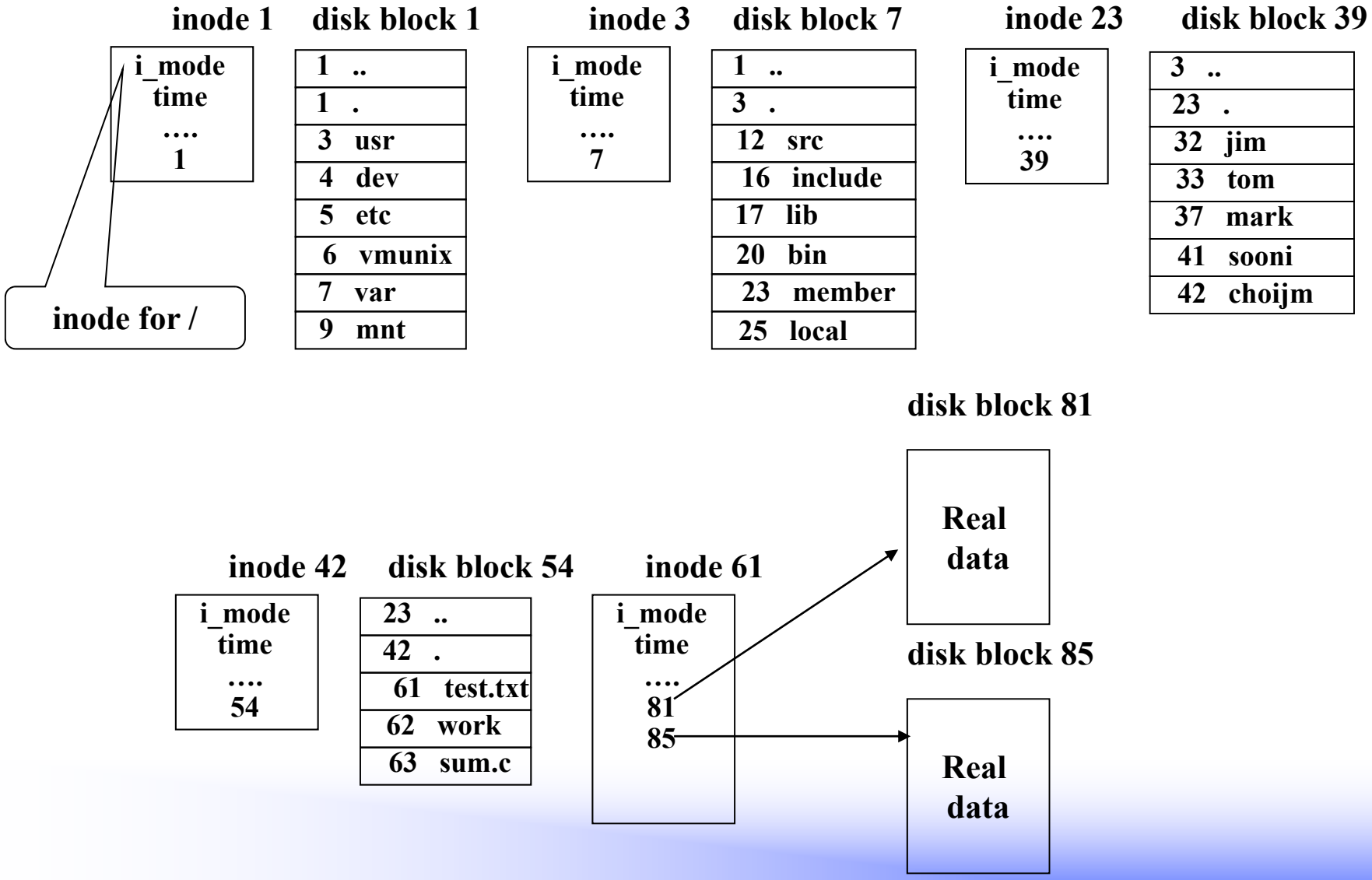
- A directory is a list of **an unique inode number** and a file/directory **name**
- An inode number represents an inode structure which contains file or directory's stat info (uid, gid, permission, size, date, ...) and address to actual data blocks
- A directory contains a current directory(`.`) and a parent directory(`..`).
- A directory can hold other directory as a subdirectory

Directory 구조

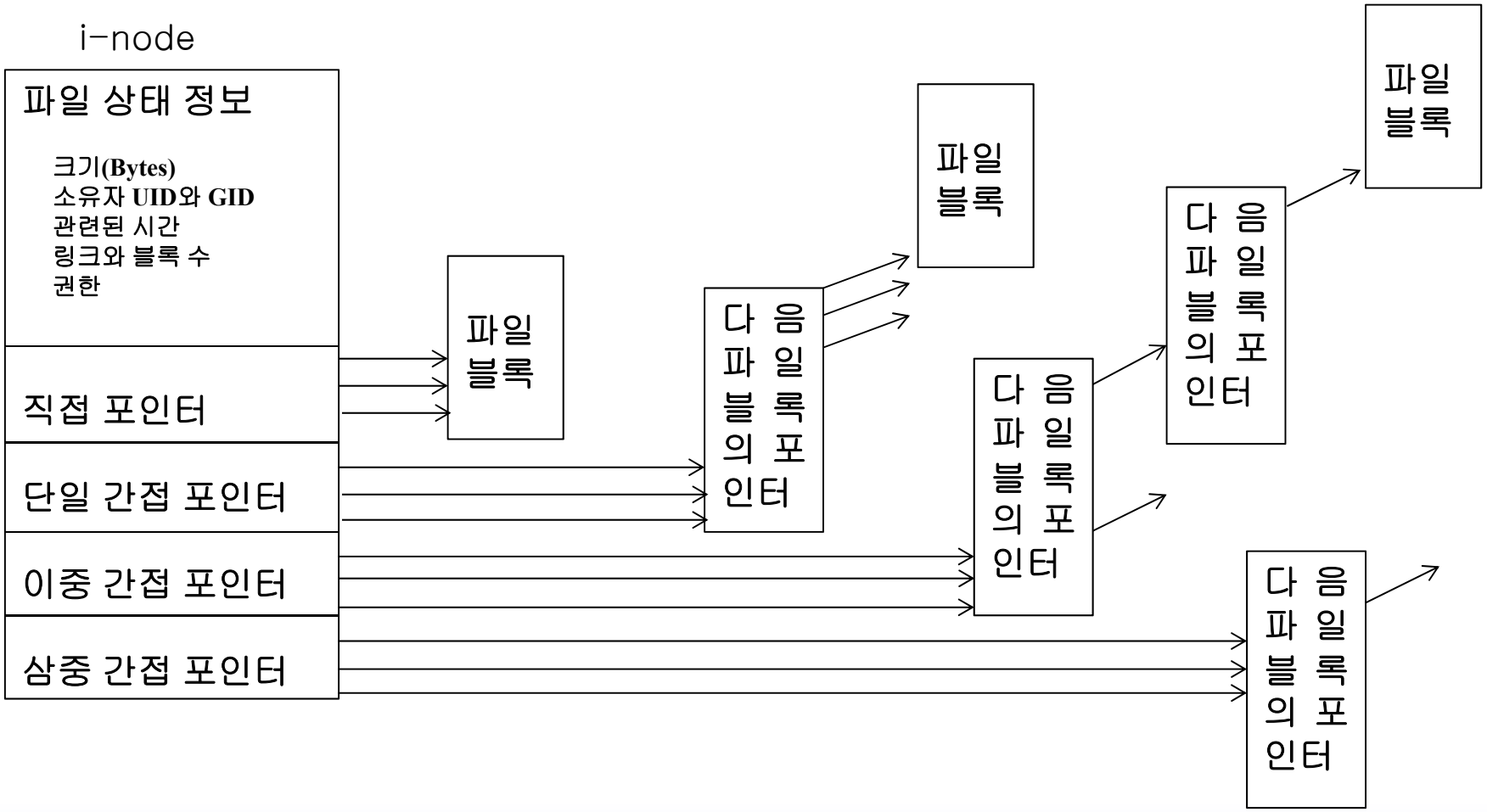
- 파일 계층 구조 (hierarchical structure)



Directory 구조



유닉스 파일 구조


i-node
파일 상태 정보
크기(Bytes)
소유자 UID와 GID
관련된 시간
링크와 블록 수
권한
직접 포인터
단일 간접 포인터
이중 간접 포인터
삼중 간접 포인터
파일 블록
다음 파일 블록의 포인터
파일 블록
다음 파일 블록의 포인터
다음 파일 블록의 포인터
파일 블록
다음 파일 블록의 포인터

연습문제:

- inode는 128Bytes, 포인터는 4Bytes, 상태정보는 68Bytes라고 가정하자. 블록의 크기가 8K Bytes이고 블록 포인터는 32bits라면:
  1. inode는 직접 포인터를 사용하여 몇 개의 파일 블록을 가질까?
  2. 직접포인터로 표시할 수 있는 파일의 크기는?
  3. 단일 간접포인터의 경우는?
  4. 이중 간접포인터, 삼중 간접포인터의 경우는?

답:

1. 단일, 이중, 삼중 간접포인터는 각각 4 Bytes이다. 그래서 128-68-(4 + 4 + 4) = 48 Bytes가 직접 포인터로 사용될 수 있는데 각 포인터는 4Bytes 씩이므로 12개의 직접 포인터가 사용 가능하다.

2. 단일 직접 포인터로 8KB, (8192Bytes), x 12 = 98,304Bytes의 파일 표시 가능

3. 단일 간접포인터용 포인터 블록으로 8KB 크기라면 블록에는 8192/4 = 2048개의 데이터 블록을 가리킬 수 있는 포인터가 있을 수 있다. 따라서 추가로 2048 x 8192 = 16,777,216 = 16MBytes의 데이터를 표시 가능

4. 이중 간접포인터는 2048 x 2048 x 8192 = 32 GB

5. 삼중 간접포인터는 2048^3 의 포인터 제공 가능

Directory Permissions

- Directory permission consists of read/write/execute.
- Directory permission is interpreted differently from file permission.
  - ✓ Read – One can list file name or subdirectory name within the directory. (This does not mean one can read files)
  - ✓ Write – One can create new files and remove existing files within the directory. (This does not mean one can modify files)
  - ✓ Execute (search) – One can move into this directory using **`cd`** command or **`chdir`** system call. (To open a file or execute a file, one should have execute permissions on all directories leading to the file)

디렉토리 관련 함수[1]

- 디렉토리 생성: mkdir(2)

```
#include <sys/types.h>
#include <sys/stat.h>
int mkdir(const char *path, mode_t mode);
```

  - ✓ path에 지정한 디렉토리를 mode 권한에 따라 생성

- 디렉토리 삭제: rmdir(2)

```
#include <unistd.h>
int rmdir(const char *path);
```

- 디렉토리명 변경: rename(2)

```
#include <stdio.h>
int rename(const char *old, const char *new);
```

디렉토리 생성/삭제/이름 변경하기

```
01  #include <sys/stat.h>
02  #include <unistd.h>
03  #include <stdlib.h>
04  #include <stdio.h>
05
06  int main(void) {
07      if (mkdir("han", 0755) == -1) {
08          perror("han");
09          exit(1);
10      }
11
12      if (mkdir("bit", 0755) == -1) {
13          perror("bit");
14          exit(1);
15      }
16
17      if (rename("han", "hanbit") == -1) {
18          perror("hanbit");
19          exit(1);
20      }
21
22      if (rmdir("bit") == -1) {
23          perror("bit");
24          exit(1);
25      }
26
27      return 0;
28  }
```

han -> hanbit로 변경

bit는 생성했다 삭제

```
# ex3_13.out
# ls -l
drwxr-xr-x   2 root other 512  1월 12일   18:06 hanbit
```

Directory의 생성 및 제거

```
#include <sys/types.h>
#include <sys/stat.h>
int mkdir(const char *pathname, mode_t mode);
```

```
#include <unistd.h>
int rmdir(const char*pathname);
```

- Return of `mkdir`:
  - ✓ Success: 0
  - ✓ Fail: -1
- Return of `rmdir`:
  - ✓ Success: 0
  - ✓ Fail: -1

Current Working Directory

```
#include <unistd.h>
int chdir(const char *path);
char *getcwd(char *name, size_t size);
```

- Return of **chdir**:
  - ✓ Success: 0
  - ✓ Fail: **-1**

- **name:** place where the current directory name is copied into
- Return of **getcwd**:
  - ✓ Success: **name**
  - ✓ Fail: **NULL**

Current Working Directory 예: my_pwd

```
/* my_pwd -- 작업 디렉토리를 프린트한다.*/

#include <stdio.h>
#include <unistd.h>
#define VERYBIG 200

void my_pwd (void);

main()
{
  my_pwd();
}

void my_pwd (void)
{
  char dirname[VERYBIG];

  if ( getcwd(dirname, VERYBIG) == NULL)
     perror("getcwd error");
  else
     printf("%s\n", dirname);
}
```

실습:

- 실습 프로그램 코드
  - ~/ACS30021/chap05/dir/mkdir/mkrm.c
    - mkdir (), chdir (), rmdir ()
  - ~/ACS30021/chap05/dir/dirinfo/getdirinfo.c
    - stat ()
  - ~/ACS30021/chap05/dir/stat/stat.c
    - stat () 와 mkdir ()

- ~/ACS30021/chap05/pilot/a/ 디렉터리에 test.txt파일 생성하여 그 내용으로,

  한국산업기술대학교

  컴퓨터 공학과

  홍길동

프로그램 코드

```
#include <stdio.h>

int main (int argc, char **argv)
{
  int status;
  FILE *fto;
  char buf[512];

  if (argc != 2) {
    fprintf(stderr, "USAGE: filename option₩n");
    return -1;
  }

  status = mkdir ("test", 0755);
  (!status) ? printf ("Directory created.₩n") :
      printf ("Unable to create directory.₩n");

  status = mkdir ("./test/testa", 0755);
  (!status) ? printf ("Directory created.₩n") :
      printf ("Unable to create directory.₩n");

  sprintf(buf, "ls -al");
  system (buf);
```

```
  status = rmdir ("test");
  (!status) ? printf ("Directory deleted.\n") :
              printf ("Unable to delete the directory.\n");

  status = chdir ("./test/testa");
  (!status) ? printf ("Current Working Directory is changed.\n") :
              printf ("Unable to change to the directory.\n");

  fto = fopen (argv[1], "w+");

  strcpy(buf, "\n\tKorea Polytechnic University \
\n\tDept. of Cybernetics \n\tHong gildong");
  fwrite(buf, strlen (buf), 1, fto);

  fprintf(fto, "\n\n\tClass of Advanced Programming\n");

  fclose(fto);

  sprintf(buf, "more %s", argv[1]);
  system (buf);

  status = chdir ("../..");
  (!status) ? printf ("Current Working Directory is changed.\n") :
              printf ("Unable to change to the directory.\n");

  sprintf (buf, "rm -rf %s", "test"); // unlink() 사용 가능
  system(buf);

  return 0;
}
```

Directory의 열기 및 닫기

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *dirname);
```

```
#include <dirent.h>
int closedir(DIR *dirptr);
```

- DIR 포인터
  - ✓ 디렉터리에 관한 정보
  - ✓ <dirent.h> 에 정의
- Return of opendir:
  - ✓ Success: **DIR ***
  - ✓ Fail: **NULL**
- Return of closedir:
  - ✓ Success: 0
  - ✓ Fail: -1

Directory의 열기 및 닫기

```
#include <stdlib.h>
#include <dirent.h>

main()
{
  DIR *dp;

  if ((dp = opendir("/tmp/dir1")) == NULL)  {
     fprintf (stderr, "Error on opening directory /tmp/dir1\n");
     exit (1);
  }

   /* 디렉토리에 대한 코드를 처리한다. */

  .
  .
  closedir(dp);
}
```

Directory 읽기

```
#include <sys/types.h>
#include <dirent.h>
struct dirent *readdir(DIR *dirptr);
void rewinddir(DIR *dirptr);
int getdents (unsigned int fd, struct ldirents *dentp,
              unsigned int count)
```

- Return of **readdir**:
  - ✓ Success: `struct dirent *`
  - ✓ Fail (if no directory entry): `NULL`
- 최신 개정된 함수:
  - ✓ `getdents (unsigned int fd, struct ldirents *dentp, unsigned int count)`
- Return of **rewinddir**: 디렉터리 offset값을 처음으로 reset 기능
  - ✓ None
- `struct dirent` consists of **d_ino** and **d_name**.

Directory 예: my_double_ls

```
#include <dirent.h>

int my_double_ls (const char *name)
{
  struct dirent *d;
  DIR *dp;

  /* 디렉토리를 개방하고, 실패여부를 점검함 */
  if ((dp=opendir(name)) == NULL)
  return (-1);

  /* 디렉토리를 살피면서 루프를 계속한다. 이때 inode 번호가 유효하면 디렉토리항을 프린트한다. */
  while (d = readdir(dp)) {
  if (d->d_ino !=0)
        printf ("%s\n", d->d_name);
  }
  /* 이제 디렉토리의 시작으로 되돌아간다 ... */
  rewinddir(dp);
  /* ... 그리고 디렉토리를 다시 프린트한다. */
  while (d = readdir(dp))  {
  if (d->d_ino != 0)
        printf ("%s\n", d->d_name);
  }
  closedir (dp);
  return (0);
}
```

Directory 예: find_entry(1)

```
#include <stdio.h> /* NULL을 정의 */
#include <dirent.h>
#include <string.h> /* 스트링 함수를 정의 */

int match(const char *, const char *);

char *find_entry(char *dirname, char *suffix, int cont)
{
  static DIR *dp=NULL;
  struct dirent *d;

  if (dp == NULL || cont == 0){
    if (dp != NULL}
        closedir (dp);
    if ((dp = opendir (dirname)) == NULL)
        return (NULL);
  }
```

Directory 예: find_entry(2)

```
 while (d = readdir(dp)) {
   if (d->d_ino == 0)
       continue;
   if (match (d->d_name, suffix))
       return (d->d_name);
  }
  closedir (dp);
  dp = NULL;
  return (NULL);
}

int match (const char *s1, const char *s2)
{
  int diff = strlen(s1)- strlen(s2);

  if (strlen(s1) > strlen(s2))
    return (strcmp(&s1[diff], s2) == 0);
  else
    return (0);
}
```

getdents () 활용 예 – 디렉토리 엔트리를 가져옴

```
#include <dirent.h>     /* Defines DT_* constants */
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/syscall.h>
#define handle_error(msg) do { perror(msg); exit(EXIT_FAILURE); } while (0)
struct linux_dirent {
   long           d_ino;
   off_t          d_off;
   unsigned short d_reclen;
   char           d_name[];
};
#define BUF_SIZE 1024

int main (int argc, char *argv[])
{
   int fd, nread;
   char buf[BUF_SIZE];
   struct linux_dirent *d;
   int bpos;
   char d_type;
```

getdents () 활용 예

```
  fd = open (argc > 1 ? argv[1] : ".", O_RDONLY | O_DIRECTORY);
  if (fd == -1) handle_error("open");

  for ( ; ; ) {
     nread = syscall (SYS_getdents, fd, buf, BUF_SIZE);
     if (nread == -1) handle_error("getdents");
     if (nread == 0) break;
     printf("---------------- nread=%d ----------------₩n", nread);
     printf("i-node#  file type  d_reclen  d_off   d_name₩n");
     for (bpos = 0; bpos < nread;) {
        d = (struct linux_dirent *) (buf + bpos);
        printf("%8ld  ", d->d_ino);
        d_type = *(buf + bpos + d->d_reclen - 1);
        printf ("%-10s ", (d_type == DT_REG) ?  "regular" : (d_type == DT_DIR) ?  "directory" :
                          (d_type == DT_FIFO) ? "FIFO" : (d_type == DT_SOCK) ? "socket" :
                          (d_type == DT_LNK) ?  "symlink" : (d_type == DT_BLK) ?  "block dev" :
                          (d_type == DT_CHR) ?  "char dev" : "???");
        printf ("%4d %10lld  %s₩n", d->d_reclen, (long long) d->d_off, (char *) d->d_name);
        bpos += d->d_reclen;
      }
  }
  exit (EXIT_SUCCESS);
}
```

getdents () 활용 예

```
// open a directory then read and print all entries

#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <linux/types.h>
#include <linux/dirent.h>
#include <linux/unistd.h>
extern int errno;

/* The line _syscall3(int, getdents, ...); should have
    been added to /usr/src/linux/include/linux/dirent.h
    for users to use this systemcall. However, it was
    omitted for some reason. Maybe a bug. */

_syscall3(int, getdents, uint, fd, struct dirent *, dirp, uint, count);
```

```
int main(int argc, char *argv[])
{
   int fd;
   struct dirent dent;

   if ((fd = open (argv[1], O_RDONLY)) < 0) {
      fprintf(stderr, "Failed to open %s₩n",
argv[1]);
      exit(1);
   } else {
      while (getdents(fd, &dent, sizeof(dent)) > 0) {
         printf("%s₩n", dent.d_name);
         lseek(fd, dent.d_off, SEEK_SET);
      }
   }
   close(fd);

   return 0;
}
```

Walking a Directory Tree (File Tree Walk)

```
#include <ftw.h>

int ftw(const char *path, int(*func)(), int depth);
```

- Return of `ftw`:
  - ✓ Success: 0
  - ✓ Fail: -1 or non-zero value returned by func
- `func`: a function called for each file/directory searched

```
int func (const char *name, const struct stat *sptr, int type)
{
    /* 함수의 내용 */
}
```

  - ✓ `type` : 방문하는 객체의 type
    - ➢ FTW_F file
    - ➢ FTW_D directory
    - ➢ FTW_DNR directory that could not be read
    - ➢ FTW_SL symbolic link
    - ➢ FTW_NS not symbolic link, stat이 실행될 수 없는 객체
- `depth`: # of file descriptors used

Walking a Directory Tree 예: list(1)

```
#include <sys/stat.h>
#include <ftw.h>

int list(const char *name, const struct stat *status, int type)
{
   /* 만일 stat 호출이 실패하면, 그냥 복귀한다. */
   if (type == FTW_NS)
    return 0;
   /* 아니면 객체 이름, 허가 그리고 만일 객체가 디렉토리이거나 심볼릭 링크이
  면 뒤에 "*"를 첨가한다. */
   if(type == FTW_F)
     printf("%-30s\t0%3o\n", name, status->st_mode&0777);
   else
     printf("%-30s*\t0%3o\n", name, status->st_mode&0777);

  return 0;
}
```

Walking a Directory Tree 예: list(2)

```
main (int argc, char **argv)
{
  int list(const char *, const struct stat *, int);

  if (argc == 1)
    ftw (".", list, 1);
  else
    ftw (argv[1], list, 1);
  exit (0);
}
```

nftw ()

```
/* The following program traverses the directory tree under the path named in its first command-line
     argument, or under the current directory if no argument is supplied. It displays various
     information about each file. The second command-line argument can be used to specify
     characters that control the value assigned to the flags argument when calling nftw().
 */
#define _XOPEN_SOURCE 500
#include <ftw.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>

static int display_info(const char *fpath, const struct stat *sb, int tflag, struct FTW *ftwbuf)
{
  printf("%-3s %2d %7jd   %-40s %d %s₩n",
      (tflag == FTW_D) ?   "d"   : (tflag == FTW_DNR) ? "dnr" :
      (tflag == FTW_DP) ?  "dp"  : (tflag == FTW_F) ?   "f" :
      (tflag == FTW_NS) ?  "ns"  : (tflag == FTW_SL) ?  "sl" :
      (tflag == FTW_SLN) ? "sln" : "???",
      ftwbuf->level, (intmax_t) sb->st_size, fpath, ftwbuf->base, fpath + ftwbuf->base);

  return 0;           /* To tell nftw() to continue */
}
```

```
int main (int argc, char *argv[])
{
  int flags = 0;

  if (argc > 2 && strchr(argv[2],'d') != NULL)
    flags |= FTW_DEPTH;
  if (argc > 2 && strchr(argv[2], 'p') != NULL)
    flags |= FTW_PHYS;

  if (nftw((argc < 2) ? "." : argv[1], display_info, 20, flags) == -1) {
    perror("nftw");
    exit(EXIT_FAILURE);
  }
  exit(EXIT_SUCCESS);
}
```

실습:

1. ./chap05/test/dir/ 의 하위 디렉터리인 1, 2, 3, 4, 5를 mkdir()를 사용하여 생성한다.
2. 각각의 디렉터리에 파일 이름이 같은, (예, ~/chap05/test/dir/1/abc_1.c 와 ~/chap05/test/dir/2/abc_2.c, 그리고 5/abc_5.c) 파일을 저장한다.
3. getdents() 또는 nftw()를 사용하여 각각의 디렉터리를 검색하여 “abc_xxx.c”와 같은 파일을 모두 찾는다.
4. 순서대로 연결하여(appending) 하나의 파일을 생성한다.

- 참고:
  - ✓ 예제 코드

File System 정보 – get filesystem statistics

```
#include <sys/statvfs.h>

int statvfs(const char *path, struct statvfs *buf);
int fstatvfs(int fd, struct statvfs *buf);
```

- path 또는 fd 에 의해 참조되는 파일 시스템에 대한 정보를 반환
- statvfs() returns information about a mounted filesystem.
  - ✓ path is the pathname of any file within the mounted filesystem.
  - ✓ buf is a pointer to a statvfs
- statfs (), fstatfs ()도 동일

structure “statvfs” is defined approximately as follows:

```
struct statvfs {
    unsigned long  f_bsize;    /* Filesystem block size */
    unsigned long  f_frsize;   /* Fragment size */
    fsblkcnt_t     f_blocks;   /* Size of fs in f_frsize units */
    fsblkcnt_t     f_bfree;    /* Number of free blocks */
    fsblkcnt_t     f_bavail;   /* Number of free blocks for unprivileged users */
    fsfilcnt_t     f_files;    /* Number of inodes */
    fsfilcnt_t     f_ffree;    /* Number of free inodes */
    fsfilcnt_t     f_favail;   /* Number of free inodes for unprivileged users */
    unsigned long  f_fsid;     /* Filesystem ID */
    unsigned long  f_flag;     /* Mount flags */
    unsigned long  f_namemax;  /* Maximum filename length */
};
```

Here the types fsblkcnt_t and fsfilcnt_t are defined in <sys/types.h>.

File System 정보 예: fsys

```
/* File:fsys.c        */
/* gcc -o fsys fsys.c */
/* fsys -- 화일 시스템 정보를 프린트한다. */
/* 화일 시스템 이름이 인수로 전달된다. */

#include <sys/statvfs.h>
#include <stdlib.h>
#include <stdio.h>

main (int argc, char **argv)
{
  struct statvfs buf;

  if (argc != 2) {
    fprintf (stderr, "usage: fsys filename\n");
    exit (1);
  }
  if (statvfs (argv[1], &buf) !=0) {
  fprintf (stderr, "statvfs error\n");
  exit (2);
  }
  printf ("%s:\tfree blocks %d\tfree inodes %d\n",
         argv[1], buf.f_bfree, buf.f_ffree);
  exit (0);
}
```

실습:

- ~/class/ACS30021/chap05/fsusage
- ~/class/ACS30021/chap05/fpath

실습:

- ~/class/ACS30021/chap05/dir/ftw
- ~/class/ACS30021/chap05/storage

Caching: sync and fsync

```
#include <unistd.h>

void sync(void);
int fsync(int filedes);
```

- **sync** – flush all the main memory buffers containing information about file systems to disk
- **fsync** – flush out all data and attributes associated with a particular file
- Return of **fsync**:
  - ✓ Success: 0
  - ✓ Fail: -1

실습:

- 연습문제 5

  디렉터리명을 명령행 인자로 입력 받아 디렉터리를 생성하고, 작업 디렉터리를 새로 생성한 디렉터리로 이동시키는 프로그램을 작성하시오.

- 연습문제 7

  현재 디렉터리의 한 파일 또는 파일들을 마지막에 지정한 디렉터리로 이동시키는 프로그램을 작성하시오.

  실행 예,

  $ mymv a.out a.dat ./dir1/

- 연습문제 8

  지정한 디렉터리를 명령행 인자로 입력 받아 파일들과 디렉터리들의 정보를 출력하는 프로그램을 작성하시오.

  실행 예,

  $ mylist ./