Programming with UNIX File Systems

(Chap 3, 4. in the book "Advanced Programming in the UNIX Environment")

Acknowledgement : Prof. Y. Moon at Kangwon Nat'l Univ.

File

- A file is a contiguous sequence of bytes.
- No format imposed by the operating system.
- Each byte is individually addressable in a disk file.

File Descriptor

- **`open()`** returns an **`fd`**, an integer value.
- Used in subsequent I/O operations on that file.
- **`close(fd)`** closes that file described by **`fd`**.
- All of a process's open files are automatically closed when it terminates.

File Descriptor

- file descriptor: 0 ~ N (N = 19? or more) // unistd.h

Value	Meaning
0	standard input
1	standard output
2	standard error
3 .. 19	fds for users

System call for file I/O

- `open()`
- `creat()`
- `close()`
- `read()`
- `write()`
- `lseek()`

open()

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open (const char *pathname, int oflag, [ mode_t  mode ]);
```

- Open a file
  - The ***open*** API is used to create new files and also to establish a connection between a process and a file
- A new file can be created if there is no such file
  - pathname : file name
  - mode : access permission. Can be omitted
  - return value : file descriptor value. if fail, return -1

open()

- Second parameter o_flag is (logical) OR of following constants
  - Ex)
```
int fd;
fd = open("afile", O_RDWR | O_CREAT, 0644);
```
  - At least one of following constants need to be specified
    - O_RDONLY : for read only. No write can be performed.
    - O_WRONLY : for write only. No read can be performed.
    - O_RDWR : for read or write, or both

open()

- Following constants for o_flag is optional
  - O_APPEND : append to the end of file on each write.
  - O_CREAT : create the file if it doesn't exist. (mode is applied)
  - O_EXCL : generate an error if O_CREAT is also specified and the file already exists.
  - O_TRUNC : if the file exists, and if the file is successfully opened for either write-only or read-write, truncate its length to 0.
  - O_SYNC : Have each write wait for physical I/O to complete

umask ( ) :

- The umask function set up default permissions for newly created files.
- Syntax:

```
#include <sys/stat.h>
mode_t umask(mode_t cmask);
```

The file mode creation mask is used whenever the process creates a new file or a new directory.

chown command is used to change the file permissions. These permissions read, write and execute permission for owner, group, and others.

Syntax

#include<unistd.h>

int chown(const char * pathname , uid_t owner , gid_t group);

- The first optional parameter indicates who – this can be (u)ser, (g)roup, (o)thers or (a)ll.
- The second optional parameter indicates opcode – this can be for adding (+), removing (-) or assigning (=) a permission.
- The third optional parameter indicates the mode – this can be (r)ead, (w)rite, or e(x)ecute.

Example: chown owner-user file

```
  chown owner-user:owner-group file
  # ls -l demo.txt
-rw-r--r-- 1 root root 0 Aug 31 05:48 demo.txt

   # chown vivek demo.txt

  # ls -l demo.txt
 -rw-r--r-- 1 vivek root 0 Aug 31 05:48 demo.txt
```

creat()

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int creat ( const char *pathname, mode_t mode );
```

- Create a new file
  - pathname : file name
  - mode : access permission
  - Return value : file descriptor. If failed, return -1

creat()

- Following two lines are functionally the same
  - `fd = creat ( pathname, mode );`
  - `fd = open ( pathname, O_WRONLY | O_CREAT | O_TRUNC, mode);`
- mode
  - `0644` → `-rw-r--r--`
  - `0755` → `-rwxr-xr-x`
  - `0444` → `-r--r--r--`

close()

```
#include <unistd.h>

int close ( int filedesc );
```

- Close a file
- If a process is terminated, all open files will be automatically closed.

read()

```
#include <unistd.h>
ssize_t  read ( int filedes,  void *buf, size_t nbytes );
```

- Read data from a file
- **buf** : memory address to store the data to be read
- **nbytes** : the number of data bytes to be read
- Return value
  - On success, the number of data bytes read
  - If the end of file, 0
  - If failed, -1
- **size_t : unsigned int**

Example : count.c

```
#include <stdlib.h> // count.c
#include <fcntl.h>
#include <unistd.h>
#define BUFSIZE 512

int main()
{
  char buffer[BUFSIZE];
  int filedes;
  ssize_t nread;
  long total = 0;

  if ((filedes = open("afile", O_RDONLY)) == -1)
  { perror("afile"); exit(-1); }

  while( (nread = read(filedes, buffer, BUFSIZE)) > 0)
    total += nread;

  close(filedes);
  printf ("total chars in afile: %ld\n", total);
  return 0;
}
```

write()

```
#include <unistd.h>
ssize_t  write (int filedes,  const void *buf, size_t nbytes);
```

- Write data to a file
- **buf** : memory address to write data
- **nbytes** : number of data bytes to write
- Return value
  - On success : number of bytes written
  - -1 if failed.

lseek()

```
#include <sys/types.h>
#include <unistd.h>
off_t  lseek (int  filedes,  off_t  offset, int  whence );
```

- Move current file offset
- Parameters
  - **Whence** : start point to move from
    - SEEK_SET
    - SEEK_CUR
    - SEEK_END
  - **Offset** : the number of bytes to move from whence (negative value is possible)
  - Return value : new file offset if OK, -1 on error

lseek()

- Current file offset
  - File read and write are performed at the point of current file offset
  - When a file is opened, curent file offset is 0
  - After file read/write, file offset is automatically updated(moved).
  - We can move current file offset to an arbitrary location.

Example : lseek1.c

```
#include <unistd.h>  /* lseek1.c */
#include <fcntl.h>

char buf1[] = "abcdefghij";
char buf2[] = "ABCDEFGHIJ";

int main() {
  int fd;

  if ( (fd = creat("file.hole", 0644)) < 0)
  { perror("file.hole"); exit(-1); }

  if (write(fd, buf1, 10) != 10) /* offset now = 10 */
    perror("buf1");

  if (lseek(fd, 40, SEEK_SET) == -1) /* offset now = 40 */
    perror("lseek");

  if (write(fd, buf2, 10) != 10) /* offset now = 50 */
    perror("buf2");

  return 0;
}
```

lseek1.c

- Output “file.hole”

	0	1	2	3	4	5	6	7	8	9
0	a	b	c	d	e	f	g	h	i	j
10	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
20	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
30	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
40	A	B	C	D	E	F	G	H	I	J

Files & Directories

stat() – returns information about a file

```
#include <sys/types.h>
#include <sys/stat.h>
int  stat (const char *pathname, struct stat *buf  );
int  fstat (int filedes, struct stat *buf  );
int  lstat (const char *pathname, struct stat *buf );
```

- put information about a file into a stat type variable “buf”
- lstat() returns information about the symbolic link, not the file referenced by the symbolic link if the give file is a symbolic link.
- Return 0 on success. -1 if failed.

stat structure (1/2)

- Defined in <sys/stat.h>

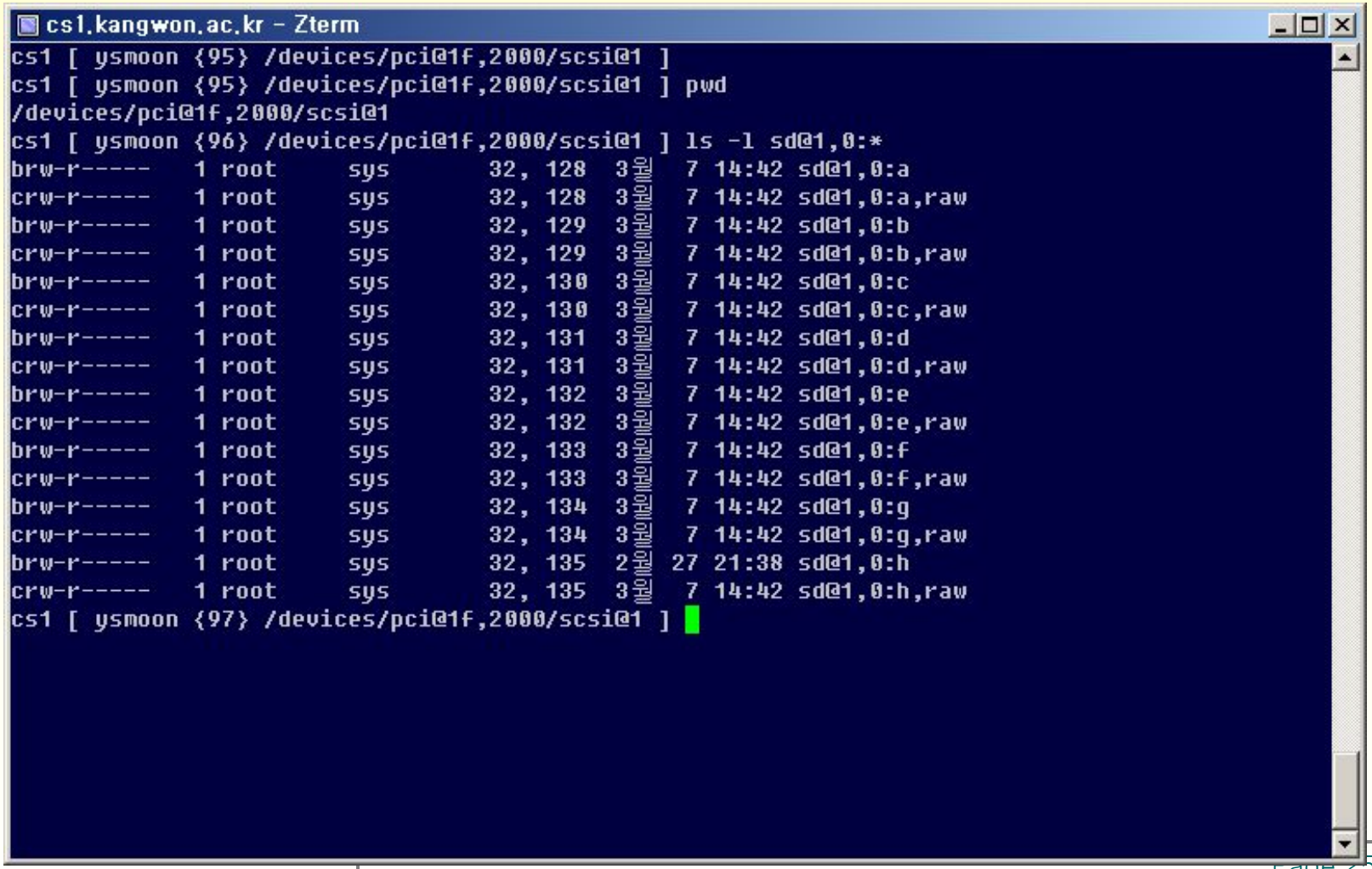
```
struct  stat {
  mode_t  st_mode;    /* file type & mode (permissions) */
  ino_t   st_ino;     /* i-node number (serial number) */
  dev_t   st_dev;     /* device number (filesystem) */
  dev_t   st_rdev;    /* device number for special files */
  nlink_t st_nlink;   /* number of links */
  uid_t   st_uid;     /* user ID of owner */
  gid_t   st_gid;     /* group ID of owner */
  off_t   st_size;    /* size in bytes, for regular files */
  time_t  st_atime;   /* time of last access */
  time_t  st_mtime;   /* time of last modification */
  time_t  st_ctime;   /* time of last file status change */
  long    st_blksize;/* best I/O block size */
  long    st_blocks; /* number of 512-byte blocks allocated */
};
```

File Types (1/3)

- Regular File
  - The most common type of file
  - Contains data of some form.
- Directory File
  - A file that contains the names of other files and pointers to information on these files.
- Character Special File
  - A type of file used for certain types of devices on a system
  - A device transmitting data character by character (`c---------`)
- Block Special File
  - Typically used for disk devices.
  - A device transmitting data block by block (`b---------`)

File Types (2/3)

- Special File

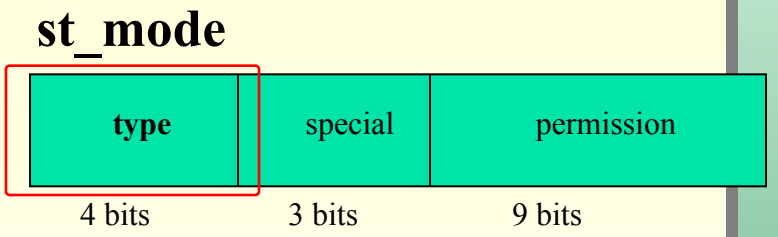


File Types (3/3)

- FIFO
  - A type of file used for interprocess communication between processes
  - Sometimes called named pipe
- socket
  - A type of file used for network communication between processes
  - Typically used in Network Programming
- Symbolic link
  - A type of file that points to another file.

Checking File Type (1/2)

- Macros for checking file type.
  - Defined in stat.h (/usr/include/sys/stat.h)
  - S_ISREG() : regular file
  - S_ISDIR() : directory file
  - S_ISCHR() : character special file
  - S_ISBLK() : block special file
  - S_ISFIFO() : pipe or FIFO
  - S_ISLNK() : symbolick link
  - S_ISSOCK() : socket
- Return 1 if YES, otherwise return 0
- Testing st_mode field in stat structure



Checking File Type (2/2)

- File type constants
  - Defined in stat.h (/usr/include/sys/stat.h)
  - S_IFREG : regular file
  - S_IFDIR : directory file
  - S_IFCHR : character special file
  - S_IFBLK : block special file
  - S_IFFIFO : pipe or FIFO
  - S_IFLNK : symbolic link
  - S_IFSOCK : socket
- S_ISxxx() macro function checks whether st_mode value has a bit of S_IFxxx constant
- Ex) `#define S_ISDIR(mode)  ((mode) & S_IFMT) == S_IFDIR)`

example: stat.c (1/2)

```
#include  <sys/types.h>   /* stat.c */
#include  <sys/stat.h>
int  main(int argc, char *argv[])
{
  int  i;
  struct stat  buf;
  char  *ptr;
  for (i = 1; i < argc; i++) {
    printf("%s: ", argv[i]);
    if (lstat(argv[i], &buf) < 0) {
      perror("lstat()");  continue;
    }
    if       (S_ISREG(buf.st_mode))  ptr = "regular";
    else if (S_ISDIR(buf.st_mode))  ptr = "directory";
    else if (S_ISCHR(buf.st_mode))  ptr = "character special";
    else if (S_ISBLK(buf.st_mode))  ptr = "block special";
    else if (S_ISFIFO(buf.st_mode)) ptr = "fifo";
    else if (S_ISLNK(buf.st_mode))  ptr = "symbolic link";
    else if (S_ISSOCK(buf.st_mode)) ptr = "socket";
    else         ptr = "** unknown mode **";
    printf("%s\n", ptr);
  }
  exit(0);
}
```

Example : stat.c (2/2)

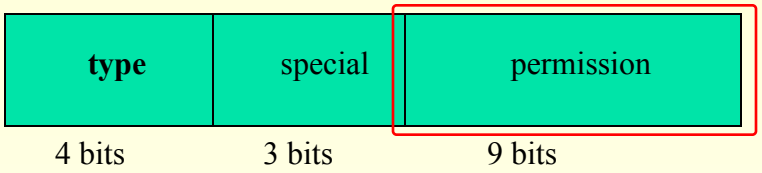
```
$ a.out  /etc  /dev/ttya  /bin  a.out
/etc: directory
/dev/ttya: symbolic link
/bin: symbolic link
a.out: regular
```

- Use lstat() to get information of a symbolic link.

File Permissions

- File access permission bits (st_mode value in stat structure)

st_mode



st_mode mask	Meaning	Octal Code
S_IRUSR	user-read	0400
S_IWUSR	user-write	0200
S_IXUSR	user-execute	0100
S_IRGRP	group-read	0040
S_IWGRP	group-write	0020
S_IXGRP	group-execute	0010
S_IROTH	other-read	0004
S_IWOTH	other-write	0002
S_IXOTH	other-execute	0001

Related UNIX Commands

- **`chmod`**
  - Set File Access Permission
  - Modify st_mode value in stat structure
- **`chown`**
  - Set User ID of file owner
  - Modify st_uid value in stat structure
- **`chgrp`**
  - Set Group ID of file owner
  - Modify st_gid in stat structure

Permissions (1/2)

- Read permission is necessary for
  - Opening a file with O_RDONLY, O_RDWR
- Write permission is necessary for
  - Opening a file with O_WRONLY, O_RDWR, O_TRUNC
- write permission and execute permission in a directory is necessary for
  - Creating a file in the directory
  - Deleting a file in the directory

Permissions (2/2)

- Whenever we want to open any type of file by name, we must have execute permission in each directory mentioned in the name, including the current directory if it is implied.
- For example, to open the file /usr/dict/words, we need execute permission in the directory /, /usr, /usr/dict, and then appropriate permission for the file words.
- In directory,
  - Read permission is necessary to read the list of files contained in the directory
  - Write permission is necessary for creating and deleting a file in the directory.
  - Execute permission is necessary for opening a file in the directory.

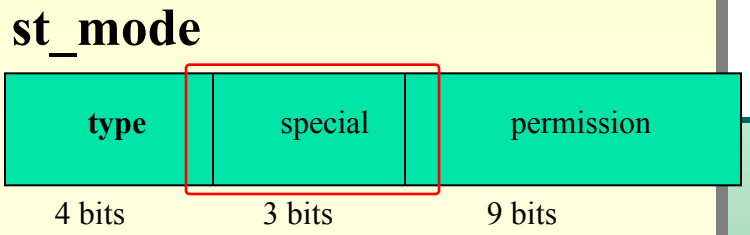
Effective User ID

- Real User ID/Real Group ID
  - Who we really are
  - Taken from our entry in the password file when we log in. normally these values don't change during a login session.
- Effective User ID/Effective Group ID
  - Process's attribute
  - Normally equals the real user ID /effective group id.
  - However, can be different when S_ISUID and S_ISGID bit is set.

  (setting the bit flag means "when this file is executed, set the effective user ID of the process to be the owner of the file (st_udi))
  - Can be understood as a user id/group id when a process is being executed.

S_ISUID , S_ISGID (1/2)

- st_mode bit in stat structure
  - S_ISUID : set-user-ID
  - S_ISGID : set-group-ID



- When executing a file where st_mode' S_ISUID bit is set
  - effective user ID of the process is set to be the owner's user id of the file
- When executing a file where st_mode's S_ISGID bit is set
  - Effective group ID of the process is set to be the owner 's group id of the file

S_ISUID , S_ISGID (2/2)

- When executing a file with S_ISUID and S_ISGID bits set
  - The file is executed with the permission of the file owner instead of the permission of Real User ID/Real Group ID
- example)

```
$ ls -al /bin/passwd
-r-sr-sr-x   1 root     sys       23500 Jul 30   2003 /bin/passwd*
```

access()

```
#include <unistd.h>

int  access (const char *pathname, int mode  );
```

- Test the permission of a file with Real User ID and Real Group ID
- Return 0 if OK, -1 on error
- mode value :

mode	Description
R_OK	test for read permission
W_OK	test for write permission
X_OK	test for execute permission
F_OK	*test for existence of tile*

example: access()

```
#include <sys/types.h> // access.c
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
  if (argc != 2) {
    printf("usage: a.out <pathname>\n");
    exit(-1);
  }

  if (access(argv[1], R_OK) < 0)
    perror("R_OK");
  else
    printf("read access OK\n");

  if (open(argv[1], O_RDONLY) < 0)
    perror("O_RDONLY");
  else
    printf("open for reading OK\n");

  return 0;
}
```

chmod(), fchmod()

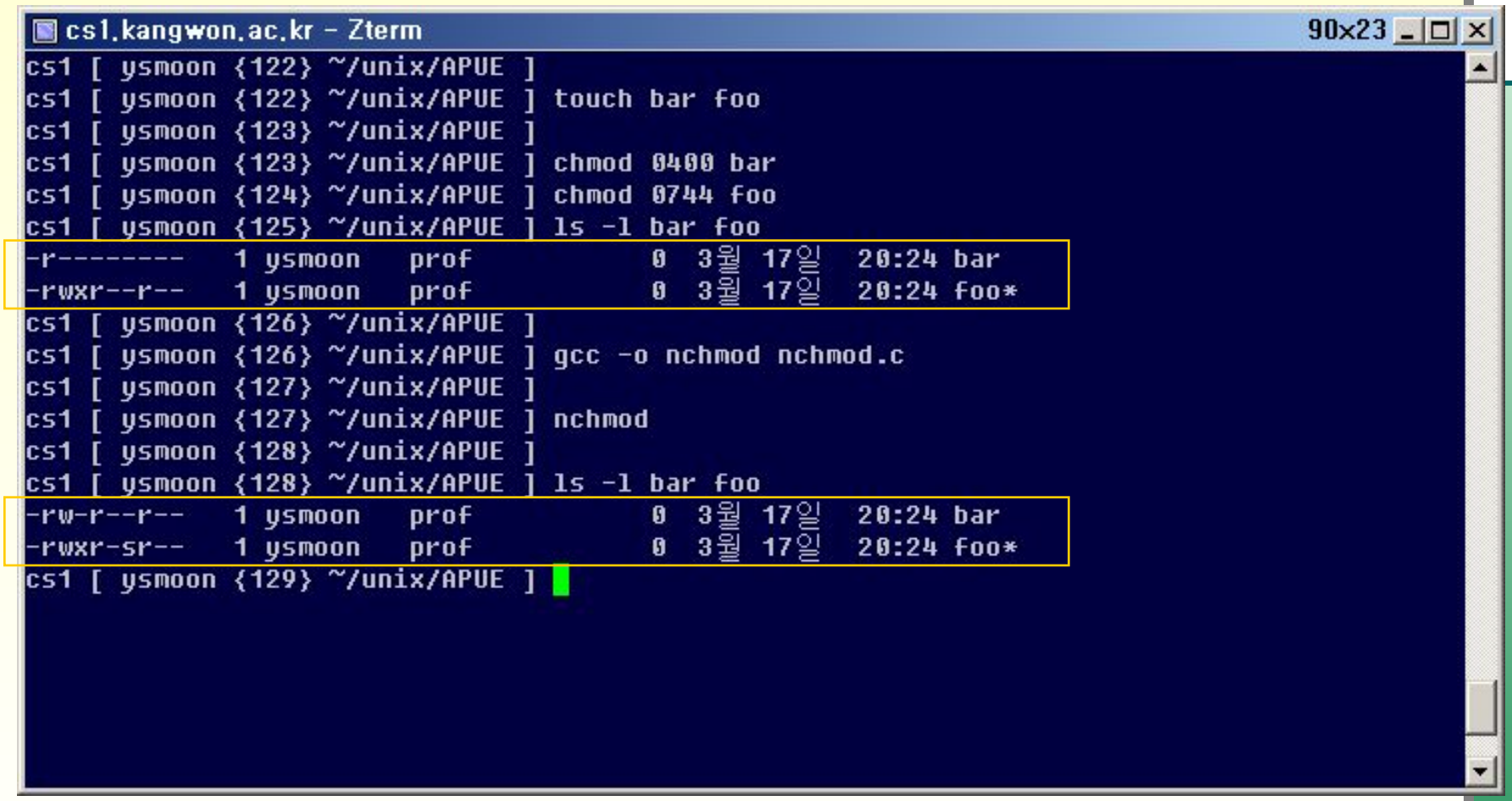
```
#include <sys/stat.h>
#include <sys/types.h>
int  chmod (const char *pathname, mode_t mode );
int  fchmod (int filedes, mode_t mode );
```

- Modify Access Permission of a File
  - Modify st_mode value of stat structure
- Return 0 if OK, -1 on error
- mode : bitwise OR
  - **S_ISUID**, **S_ISGID**, *S_ISVTX*
  - **S_IRUSR**, **S_IWUSR**, **S_IXUSR**
  - **S_IRGRP**, **S_IWGRP**, **S_IXGRP**
  - **S_IROTH**, **S_IWOTH**, **S_IXOTH**

example: chmod() (1/2)

```
#include <sys/types.h> // nchmod.c
#include <sys/stat.h>

int main()
{
  struct stat    statbuf;

  /* turn on both set-group-ID and group-execute */
  if (stat("foo", &statbuf) < 0)
    perror("stat(foo)");
  if (chmod("foo", (statbuf.st_mode | S_IXGRP | S_ISGID)) < 0)
    perror("chmod(foo)");

  /* set absolute mode to "rw-r--r--" */
  if (chmod("bar", S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH) < 0)
    perror("chmod(bar)");

  return 0;
}
```

example: chmod() (2/2)



chown()

```
#include <sys/types.h>
#include <unistd.h>
int  chown (const char *pathname, uid_t owner, gid_t group );
int  fchown (int filedes, uid_t owner, gid_t group );
int  lchown (const char *pathname, uid_t owner, gid_t group );
```

- Modify User ID and Group ID of a file
  - Modify st_uid, st_gid value in stat structure
- Return 0 if OK, -1 on error
- lchown() modifies symbolic link itself
- Can be different in different versions of UNIX
  - On BSD-based system , only super-user can modify
  - On System V UNIX sysem, general user can modify.

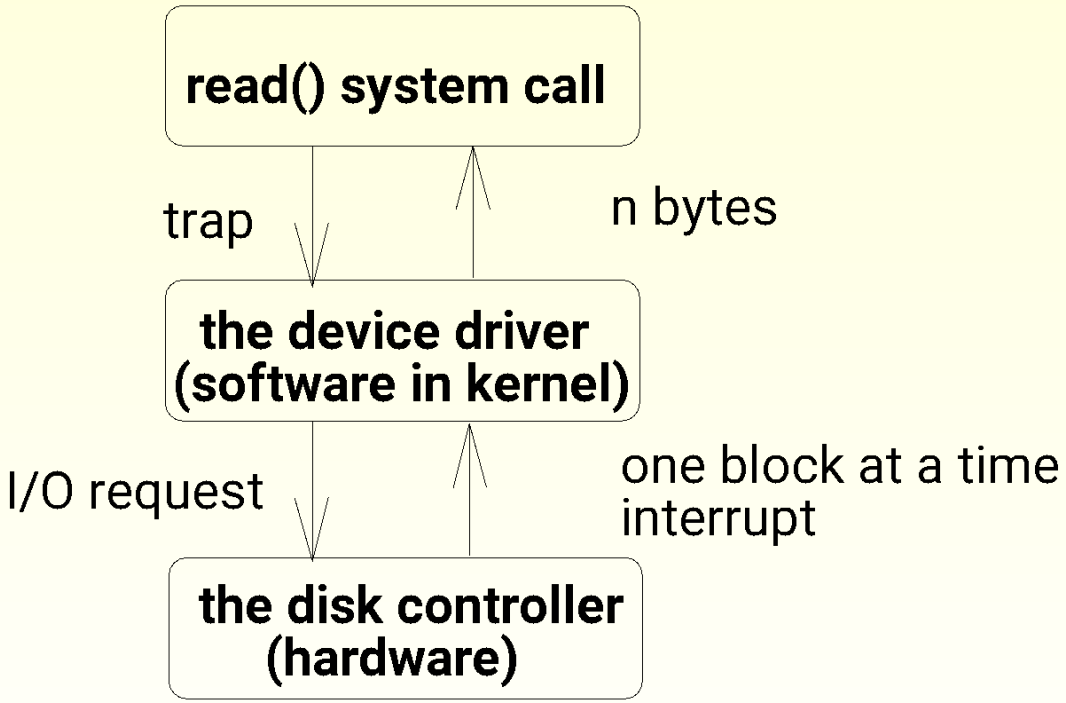
truncate(), ftruncate()

```
#include <sys/types.h>
#include <unistd.h>

int truncate (const char *pathname, off_t length  );
int ftruncate (int filedes, off_t length  );
```

- Reduce the size of a file into “length”
- Return 0 if OK, -1 on error

Block I/O

- I/O is always done in terms of blocks.
- Sequence of a read() system call



link()

```
#include <unistd.h>

int  link (const char *existingpath, const char* newpath);
```

- Any file can have multiple directory entries pointing to its i-node.
- This function creates a link to an existing file
- Increment the link count

unlink()

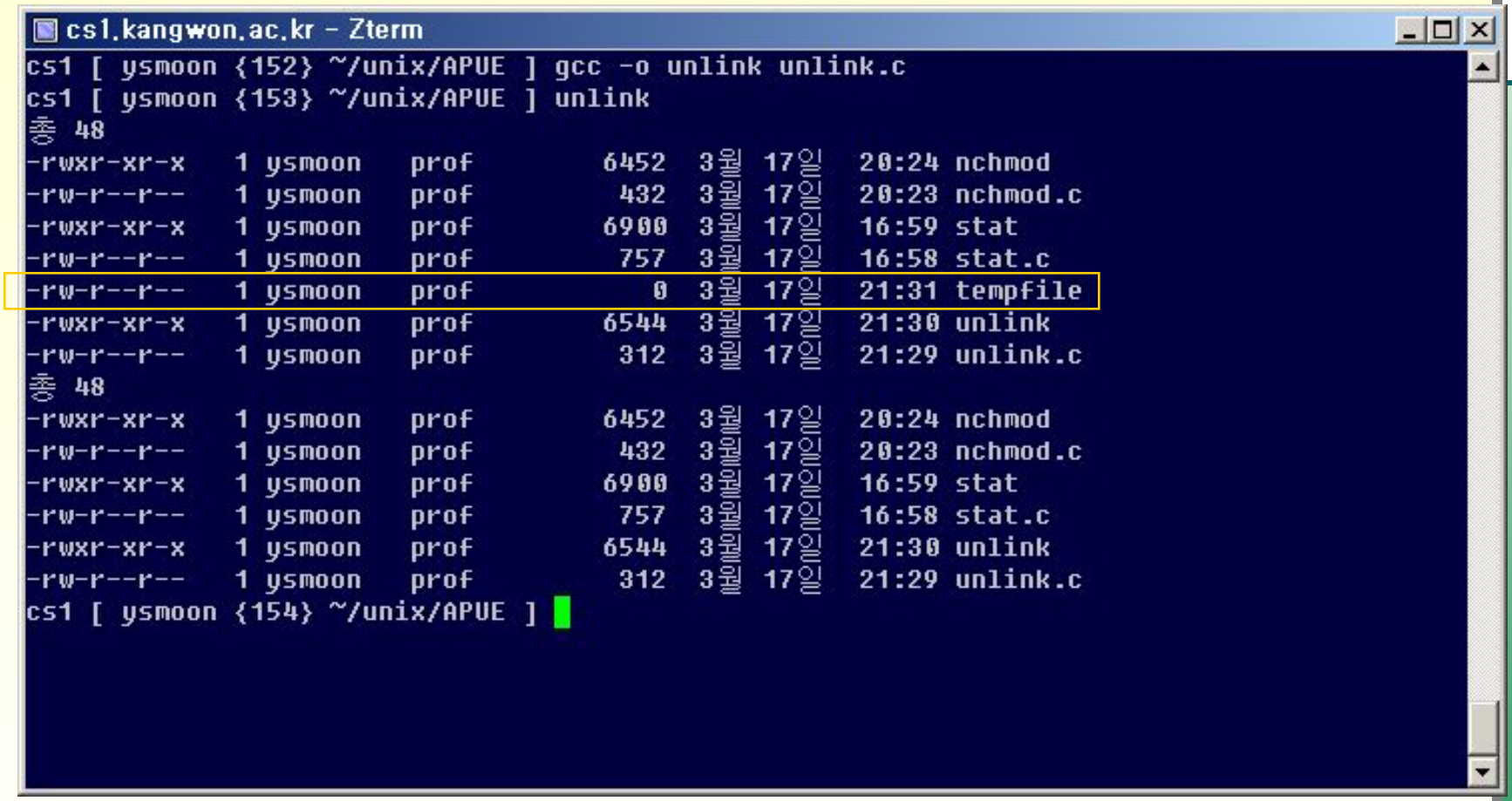
```
#include <unistd.h>

int  unlink (const char *pathname);
```

- Removes the directory entry and decrements the link count of the file.
- Return 0 if OK, -1 on error
- If the file is deleted, inode and data block become freed (deleted)

example: unlink() (1/2)

```
#include <sys/types.h> // unlink.c
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main() {
  int fd, len;
  char buf[20];
  fd = open("tempfile", O_RDWR | O_CREAT | O_TRUNC, 0666);
  if (fd == -1) perror("open1");
  close(fd);

  system("ls -l");

  unlink("tempfile");

  system("ls -l");

  return 0;
}
```

example: unlink() (2/2)



symlink() - Symbolic Link

- Symbolic Link in an indirect pointer to a file
  - Contains a path of an actual file

```
#include <unistd.h>

int  symlink (const char *actualpath, const char *sympath );
```

- Create a Symbolic Link
  - The data of Symbolic Link File becomes actualpath
  - The size of Symbolic Link File becomes the size of actualpath string
- Returns 0 if OK, -1 on error

File Times

- st_atime in stat structure
  - Last access time of file data (read)
- st_mtime in stat structure
  - Last-modification time of file data (write)
- st_ctime in stat structure
  - Last-change time of i-node status (chmod, chown)

utime()

```
#include <sys/types.h>
#include <utime.h>
int utime (const char *pathname, const struct utimbuf *times );
```

- Change the access time and the modification time of a file
- If times is NULL, set as current time.
- Return 0 if OK, -1 on error
- UNIX command : touch

```
struct utimbuf {
     time_t  actime;      /* access time */
     time_t  modtime;     /* modification time */
}
```

- Each field is a value that counts seconds since 1970-1-1 00:00

Directory

- Directory file
  - Possible to use functions such as open, read, close
  - Functions for managing a directory file are provided
- Contents of a directory file are stored as dirent type array
  - file name:
  - i-node number

```
#include <dirent.h>
struct dirent {
  ino_t d_ino;                  /* i-node number */
  char  d_name[NAME_MAX + 1];   /* filename */
}
```

Directory Access (opendir(), readdir())

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir (const char *pathname);
struct dirent *readdir(DIR *dp);
```

- opendir() : open a directory
- readdir() : read a directory contents
  - For each directory read, current file offset of the directory file is incremented by the size of dirent structure
  - For directory read, read permission of the directory is necessary.
  - Although there is write permission in directory file, write function can not be performed.

    Instead, we can use mkdir(), rmdir()
- Return the pointer address of dirent structure if OK, NULL on error

rewinddir(), closedir()

```
#include <sys/types.h>
#include <dirent.h>

void rewinddir (DIR *dp);
int closedir (DIR *dp);
```

- rewinddir()
  - Move current file offset of a directory file into the start point(first entry).
- closedir()
  - Close directory file

mkdir()

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

int  mkdir (const char *pathname, mode_t  mode );
```

- Create a new directory
- Return 0 if OK, -1 on error
- If OK, "." and .." file entries are automatically created.
  - "." points to current directory's i-node,
  - ".." points to parent directory's i-node

rmdir()

```
#include <unistd.h>

int rmdir (const char *pathname );
```

- Remove an empty directory.
- Return 0 if OK, -1 on error

example: listfiles.c (1/5)

```
#include <sys/types.h> // listfiles.c
#include <sys/mkdev.h>
#include <sys/stat.h>
#include <dirent.h>
#include <stdio.h>

/* typeOfFile - return the letter indicating the file type. */
char typeOfFile(mode_t mode)
{
  switch (mode & S_IFMT) {
    case S_IFREG: return('-');
    case S_IFDIR: return('d');
    case S_IFCHR: return('c');
    case S_IFBLK: return('b');
    case S_IFLNK: return('l');
    case S_IFIFO: return('p');
    case S_IFSOCK: return('s');
  }
  return('?');
}
```

example: listfiles.c (2/5)

```
/* permOfFile - return the file permissions in an "ls"-like string. */
char* permOfFile(mode_t mode) {
  int i;
  char *p;
  static char perms[10];

  p = perms;
  strcpy(perms, "---------");
  for (i=0; i < 3; i++) {
    if (mode & (S_IREAD >> i*3)) *p = 'r';
    p++;
    if (mode & (S_IWRITE >> i*3)) *p = 'w';
    p++;
    if (mode & (S_IEXEC >> i*3)) *p = 'x';
    p++;
  }
  if ((mode & S_ISUID) != 0) perms[2] = 's';
  if ((mode & S_ISGID) != 0) perms[5] = 's';

  return(perms);
}
```

example: listfiles.c (3/5)

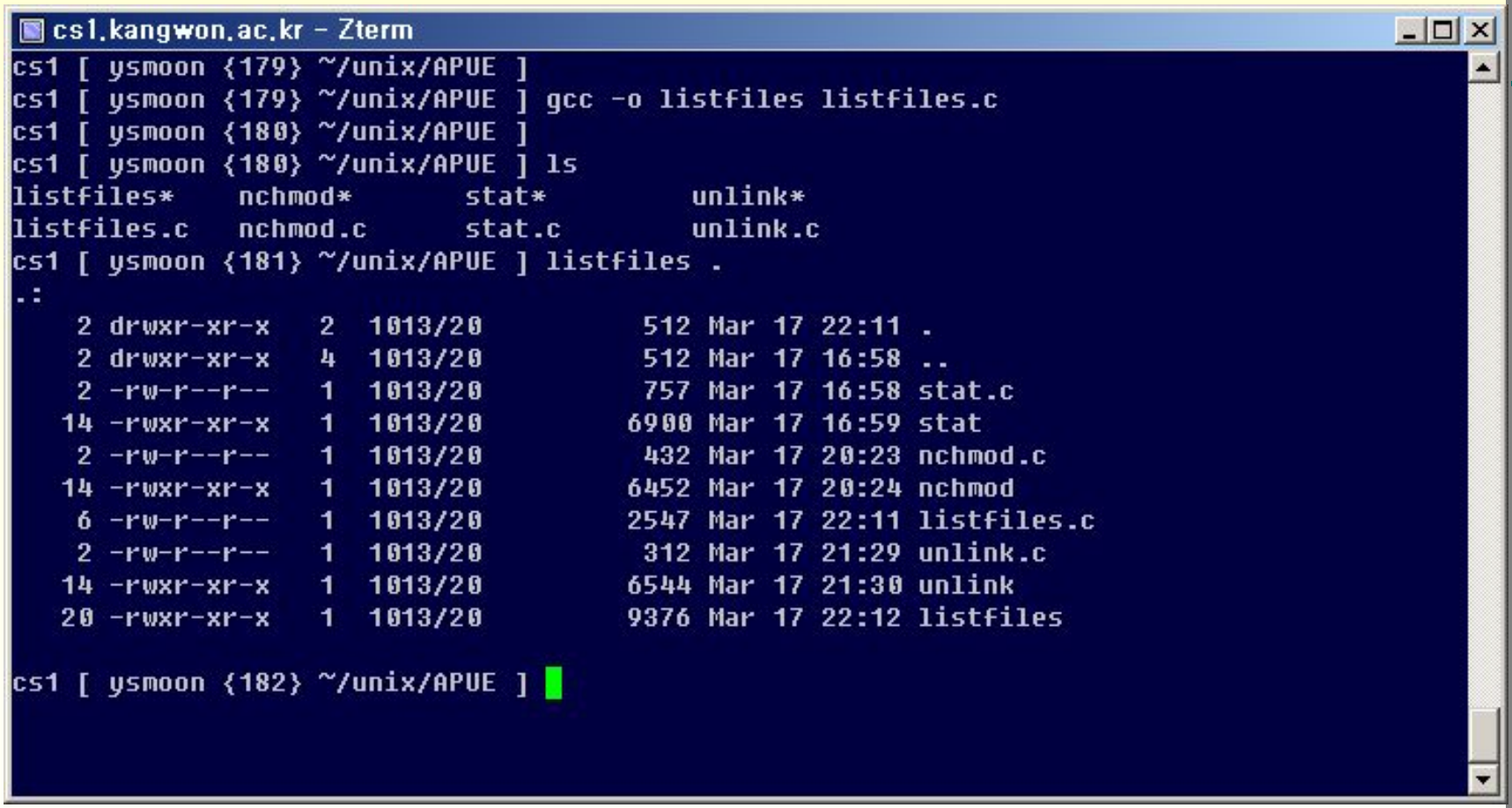
```
/* outputStatInfo - print out the contents of the stat structure. */
void outputStatInfo(char *pathname, char *filename, struct stat *st) {
  int n;
  char slink[BUFSIZ+1];

  printf("%5d ", st->st_blocks);
  printf("%c%s ", typeOfFile(st->st_mode), permOfFile(st->st_mode));
  printf("%3d ", st->st_nlink);
  printf("%5d/%-5d ", st->st_uid, st->st_gid);

  if (((st->st_mode & S_IFMT)!=S_IFCHR) && ((st->st_mode & S_IFMT)!=S_IFBLK))
    printf("%9d ", st->st_size);
  else
    printf("%4d,%4d ", major(st->st_rdev), minor(st->st_rdev));

  printf("%.12s ", ctime(&st->st_mtime) + 4);
  printf("%s", filename);
  if ((st->st_mode & S_IFMT) == S_IFLNK) {
    if ((n = readlink(pathname, slink, sizeof(slink))) < 0)
      printf(" -> ???");
    else printf(" -> %.*s", n, slink);
  }
}
```

example: listfiles.c (4/5)

```
int main(int argc, char **argv) {
  DIR *dp;
  char *dirname;
  struct stat st;
  struct dirent *d;
  char filename[BUFSIZ+1];
  /* For each directory on the command line... */
  while (--argc) {
    dirname = *++argv;
    if ((dp = opendir(dirname)) == NULL) /* Open the directory */
      perror(dirname);
    printf("%s:\n", dirname);
    while ((d = readdir(dp)) != NULL) { /* For each file in the directory... */
      /* Create the full file name. */
      sprintf(filename, "%s/%s", dirname, d->d_name);
      if (lstat(filename, &st) < 0) /* Find out about it. */
        perror(filename);
      outputStatInfo(filename, d->d_name, &st); /* Print out the info. */
      putchar('\n');
    }
    putchar('\n');
    closedir(dp);
  }
  return 0;
}
```

example: listfiles.c (5/5)

- 실행 결과



chdir(), fchdir()

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

- Change current working directory (of a process)

  (**`$cd`**)

- Return 0 if OK, -1 on error
- Current working directory (cwd) is a property of a process

getcwd()

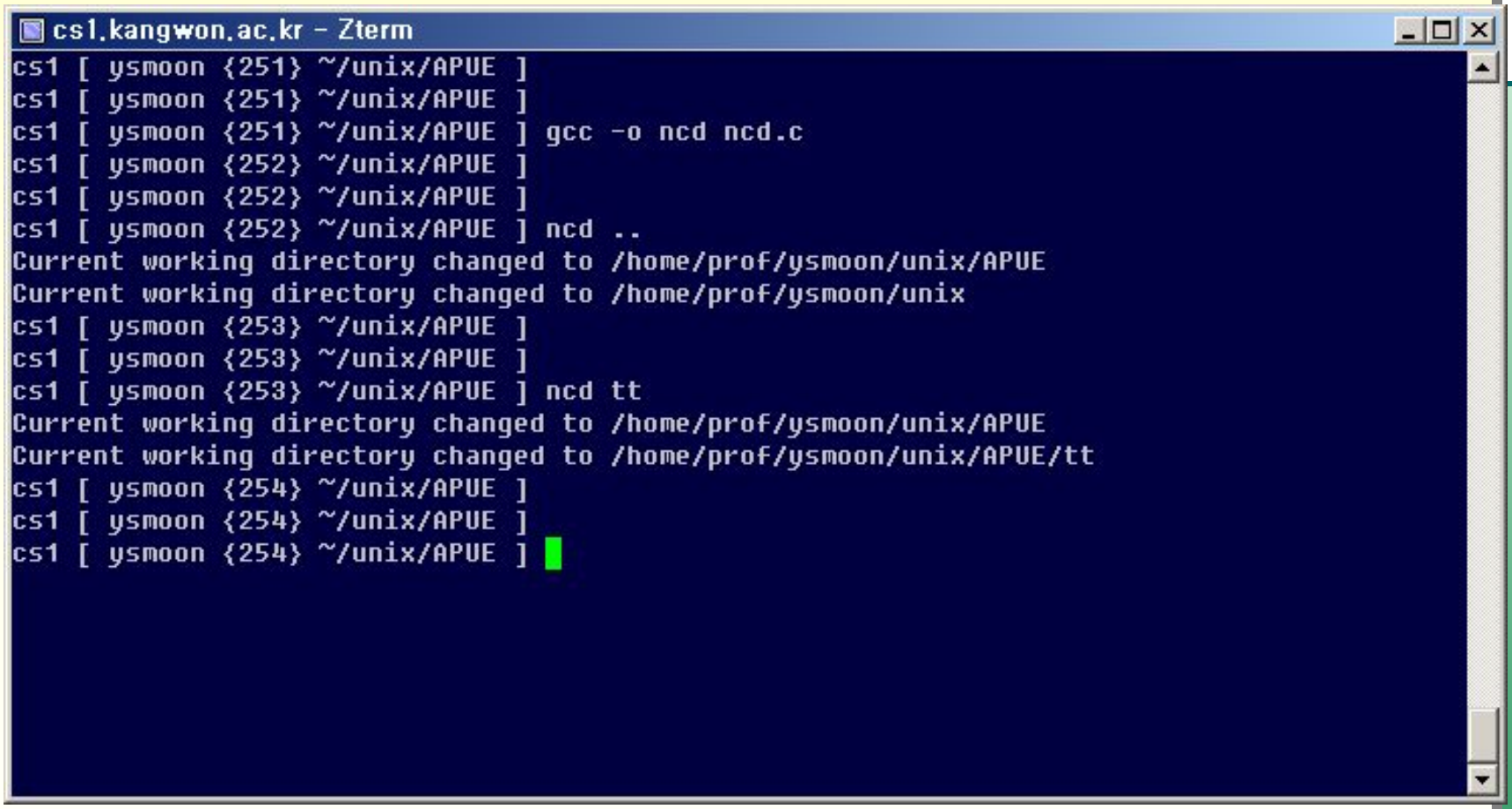
```
#include <unistd.h>

char *getcwd (char *buf, size_t size );
```

- Get the path of current working directory (**$ pwd**)
- Return the pointer address of buf, NULL if failed.

example: chdir(), getcwd() (1/2)

```
#include <unistd.h> // ncd.c
#include <stdio.h>
#define PATH_MAX 1024

int main(int argc, char **argv)
{
  char path[PATH_MAX+1];

  if (argc == 1) exit(-1);

  if (argc == 2) {
    if(getcwd(path, PATH_MAX) == NULL) perror("error getcwd");
    else printf("Current working directory changed to %s \n", path);

    if(chdir(argv[1]) < 0) perror("error chdir");
    else {
       if(getcwd(path, PATH_MAX) == NULL) perror("error getcwd");
       else printf("Current working directory changed to %s \n", path);
    }
  } else
    perror("Too many arguments");
}
```

example: chdir(), getcwd() (2/2)



sync(), fsync()

```
#include <unistd.h>

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

- Write data in all the block buffers to disk
- sync() is normally called by a system daemon process for every 30 seconds
  - this guarantees regular flushing of the kernel's block buffers.
- fsync() performs sync to a single file (specified by filedes) and waits for the I/O to complete before returning.
- **`O_SYNC`** flag (all write() with sync mode)