# 제 4 장
# I/O devices

ACS30021
고급 프로그래밍

나보균 (bkna@kpu.ac.kr)

컴퓨터 공학부
한국산업기술 대학교

# 학습 목표

- 입출력 장치의 기본
  - ✓ 파일
  - ✓ 캐릭터 장치
  - ✓ 블록 장치
- 입출력 관련 함수
  - ✓ open, read, write, close, lseek, tell, ioctl, fcntl
  - ✓ fopen, fread, fwrite, fclose, fseek, ftell
- Non-blocking device
  - ✓ select
- 파일 속성
  - ✓ stat, fstat, statfs
- Programmed I/O v.s. Memory-mapped I/O

# 입출력 장치의 기본: 파일

- 데이터 저장의 기본 단위
- 장치 구동 단위
- 디렉터리 관리
- 파일 구성 요소
  - ✓ 파일명 – 사용자가 파일에 접근 시 사용
  - ✓ inode
    - ➢ 번호로 구성되며, 파일에 관한 정보와 저장 데이터 블록의 주소 등이 포함
    - ➢ 파일의 종류, 덥근 권한, 하드링크 수, 소유자의 UID와 GID, 파일의 크기, 파일의 접근 시각/수정 시각, inode 변경 시각
  - ✓ 데이터 블록 – 실제 데이터가 저장되는 공간
- 파일의 종류
  - ✓ 일반파일
  - ✓ 특수파일(character device file, block device file)
    - ➢ 장치번호를 inode에 저장
    - ➢ 예, 솔라리스에서 기본 문자 장치는 섹터 단위(512Bytes)로 데이터 접근, 블록장치는 블록 단위(8KBytes)로 데이터 접근
  - ✓ 디렉터리
    - ➢ 연관된 데이터 블록에 해당 디렉터리 내 파일 목록과 inode 저장

# I/O blocking: Why ?

- CPU operates much faster than the disk or the network does
  - ✓ A very fast disk has 5 ms seek time
  - ✓ On a 500 MHz Pentium III machine, a task can execute about 1,250,000 assembler instructions during one seek time

# When to Block ?

- When to Block
  - ✓ Reading
    - ➢ No data has arrived yet
  - ✓ Writing
    - ➢ Internal buffers are full and waiting for transmission and your task requests more data to be sent
  - ✓ Connecting
    - ➢ accept( ) and connect( ) system calls find no pending connections in the listening queue

# Alternatives to I/O Blocking

- While waiting for a system request to finish, the task could
  - ✓ Test the integrity of its data
  - ✓ Start and track other requests
  - ✓ Wait for several socket connections
  - ✓ Process some CPU-intensive calculations

# Blocked I/O to Nonblocked I/O – select ()

- Blocked I/O
- 일정 시간 대기 후 unblocking
- GUI, Network 등에 사용

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

int select (
            int n ,                         //
            fd_set *readfds,                //
            fd_set *writefds,               //
            fd_set *exceptfds,              //
            struct timeval *timeout         //
);

FD_CLR(int fd, fd_set *set);
FD_ISSET(int fd, fd_set *set);
FD_SET(int fd, fd_set *set);
FD_ZERO(fd_set *set);
```

# Blocked I/O to Nonblocked I/O – select ()

```
MsgType *net_udp_recv_msg ( sock, . . .)
{
  . . .
  fd_set readfds;

  FD_ZERO(&readfds);

  for(;;) {

    FD_SET(sock, &readfds);

    if(select(sock + 1, &readfds, NULL, NULL, 0) < 0) {   // sock + 1
      fprintf(stderr, "select error\n");
      return(NULL);
    }

    /* Get video packet from remote if available. */
    if(FD_ISSET (sock, &readfds)) {
      . . .
      recvfrom(sock,  . . . );
      . . .

      return message;
    } else {
       /* Must be non-blocking job here */
         . . .
    }
  }
}
```

# 파일 핸들과 파일 포인터

- In Unix system, each device is regarded as a file

- file descriptor
  - ✓ int fd;
  - ✓ open (fd, ...)

- file pointer
  - ✓ FILE *fp;
  - ✓ fopen (fp, ... )
  - ✓ Data bufferring

# Open과 file 허가 오류

- open 과 파일허가 오류 번호: errno
  - ✓ EACCESS error(허가가 거부됨)
  - ✓ EEXIST error(file이 이미 존재함)
  - ✓ 예

```
fd = open (pathname, O_WRONLY|O_CREAT|O_TRUNC, 0600);
fd = open (pathname, O_WRONLY|O_CREAT|O_EXCL, 0600);
```

# File 정보의 획득: stat와 fstat

- The **stat** and **fstat** system calls

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

int stat(const char *pathname, struct stat *buf);
int fstat(int fd, struct stat *buf);
```

  - ✓ return:
    - ➢ success: 0
    - ➢ fail: −1

# File 정보의 획득: stat와 fstat

- Each file's properties are in the

```
struct stat {
    dev_t     st_dev;       /* the logical device */
    ino_t     st_ino;       /* inode number */
    mode_t    st_mode;      /* permission and file type */
    nlink_t   st_nlink;     /* # of hard links */
    uid_t     st_uid;       /* user id */
    gid_t     st_gid;       /* group id */
    dev_t     st_rdev;      /* device if file is device */
    off_t     st_size;      /* logical file size */
    time_t    st_atime;     /* last data read time */
    time_t    st_mtime;     /* last data write time */
    time_t    st_ctime;     /* last stat write time */
    long      st_blksize;   /* I/O block size */
    long      st_blocks;    /* # of physical blocks */
}
```

# File 정보의 획득: stat와 fstat

- stat와 fstat의 예

```
struct stat s;
int fd, retval;

fd = open("tmp/dina", O_RDWR);

/* s는 이제 아래의 명령이나 ... */
retval = stat("/tmp/dina", &s);

/* 또는 아래의 명령에 의해 채워질 수 있다. */
retval = fstat(fd, &s);
```

★ fxxx () 들 중 file handle을 매개변수로 한다.

✓ fcntl () 도 핸들을 매개 변수로

# stat와 fstat 예: filedata(1)

```
/* filedata -- 한 화일에 관한 정보를 출력 */

#include <stdio.h>
#include <sys/stat.h>

/* 허가 비트가 설정되어 있는지 결정하기 위해 octarray를 사용 * /
static short octarray[9] = {0400, 0200, 0100,
                            0040, 0020, 0010,
                            0004, 0002, 0001};

/* 화일 허가에 대한 기호화 코드 끝부분의 null 때문에 길이가 10문자이다. */
static char perms[10] = "rwxrwxrwx";

int filedata (const char *pathname)
{
 struct stat statbuf;
 char descrip[10];
 int j:

 if(stat (pathname, &statbuf) == -1) {
    fprintf (stderr, "Couldn't stat %s\n", pathname);
    return (-1);
 }
```

# stat와 fstat 예: filedata(2)

```
/* 허가를 읽기 가능한 형태로 바꾼다. */

 for(j=0; j<9; j++)  {
    / * 비트별 AND를 사용하여 허가가 설정되었는지 테스트 */
    if (statbuf.st_mode & octarray[j])
         descrip[j] = perms[j];
    else
         descrip[j] = '-';
 }
 descrip[9] = '\0'; /* 하나의 문자열을 가지도록 확인 */

 /* 화일 정보를 출력한다. */
 printf ("\nFile %s :\n", pathname);
 printf ("Size %ld bytes\n", statbuf.st_size);
 printf(“Uid %d, Gid %d\n\n", statbuf.st_uid, statbuf.st_gid);
 printf ("Permissions: %s\n", descrip);

 return (0);
}
```

# stat와 fstat 예: lookout(1)

```
/* lookout -- 화일이 변경될 때 메시지를 프린트 */
    #include <stdlib.h>
    #include <stdio.h>
    #include <sys/stat.h>

    #define MFILE    10

    void cmp(const char *, time_t);
    struct stat sb;

    main (int argc, char **argv)
    {
      int j;
      time_t last_time[MFILE+1];

      if(argc < 2) {
         fprintf (stderr, "usage: lookout filename ...\n");
        exit (1);
      }
      if(argc > MFILE) {
       fprintf (stderr, "lookout: too many filenames\n");
       exit (1);
      }
```

# stat와 fstat 예: lookout(2)

```
/* 초기화 */
 for (j=1; j<=argc; j++) {
   if (stat (argv[j], &sb) == -1) {
     fprintf (stderr, "lookout: couldn't stat %s\n", argv[j]);
     exit (1);
   }
   last_time[j] = sb.st_mtime;
 }

 /* 화일이 변경될 때까지 루프 */
 for (;;) {
   for (j=1; j<=argc; j++)
     cmp (argv[j], last_time[j]);

   sleep (60); /* 60초간 쉰다. */
 }
}
```

# stat와 fstat 예: lookout(3)

```
void cmp(const char *name, time_t last)
{
 /* 파일에 관한 통계를 읽을 수 있는 한 변경시간을 검사한다. */
 if (stat(name, &sb) == 1 || sb.st_mtime ! = last) {
   fprintf (stderr, "lookout: %s changed\n", name);
   exit (0);
 }
}
```

# stat와 fstat 예: addx

```
/* addx -- 화일에 수행허가를 추가 */
#include <stdlib.h>
#include <stdio.h>
#include <sys/stat.h>
#define XPERM  0100        / *  소유자에 대한 수행 허가 */

main(int argc, char **argv)
{
 int k; struct stat statbuf;

 /* 인수 리스트의 모든 화일에 대해 루프 */
 for (k=1; k<argc; k++){
   if (stat (argv[k], &statbuf) == -1) /* 현행 화일 모드를 얻음 */
      fprintf (stderr, "addx: couldn't stat %s\n", argv[k]);
   continue;

   statbuf.st_mode |= XPERM; /* 비트별 OR 연산을 사용하여 수행허가의 추가를 시도 */
   if (chmod (argv[k], statbuf.st_mode) == -1)
     fprintf (stderr, "addx: couldn't change mode for %s\n", argv[k]);
 }
 exit (0);
}
```

# 파일 접근 권한 검색

- 함수를 사용한 **파일 접근 권한 검색** : access(2)

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

  - ✓ path에 지정된 파일이 amode로 지정한 권한을 가졌는지 확인하고 리턴
  - ✓ 접근권한이 있으면 0을, 오류가 있으면 -1을 리턴
  - ✓ 오류메시지
    - ➢ ENOENT : 파일이 없음
    - ➢ EACCESS : 접근권한이 없음
  - ✓ mode 값
    - ➢ R_OK : 읽기 권한 확인
    - ➢ W_OK : 쓰기 권한 확인
    - ➢ X_OK : 실행 권한 확인
    - ➢ F_OK : 파일이 존재하는지 확인

# access 함수를 이용해 접근 권한 검색하기

```
01  #include <sys/errno.h>
02  #include <unistd.h>
03  #include <stdio.h>
04
05  extern int errno;
06
07  int main(void) {
08      int per;
09
10      if (access("unix.bak", F_OK) == -1 && errno == ENOENT)
11          printf("unix.bak: File not exist.\n");
12
13      per = access("unix.txt", R_OK);
14      if (per == 0)
15          printf("unix.txt: Read permission is permitted.\n");
16      else if (per == -1 && errno == EACCES)
17         printf("unix.txt: Read permission is not permitted.\n");
18
19      return 0;
20  }
```

```
# ls -l unix*
-rw-r--r--   1 root other 24  1월   8일   15:47 unix.txt
# ex3_6.out
unix.bak: File not exist.
unix.txt: Read permission is permitted.
```

# Changing Permission and Ownership

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
int chmod(const char *pathname, mode_t newmode);
int fchmod(int fd, mode_t mode);
int chown(const char *pathname, uid_t uid, gid_t gid);
```

- `newmode`: permission mode
- `uid`: user id
- `gid`: group id
- return:
  - ✓ success: 0
  - ✓ fail: -1
- file의 소유자나 super user만 사용 가능

# 파일링크

- 링크
  - ✓ 기존 파일이나 디렉토리에 접근할 수 있는 새로운 이름
  - ✓ 같은 파일/디렉토리지만 여러 이름으로 접근 가능
  - ✓ 하드링크 : 기존 파일과 동일한 inode 사용, inode에 저장된 링크 개수 증가
  - ✓ 심볼릭 링크 : 기존 파일에 접근하는 다른 파일 생성(다른 inode 사용)
- 하드링크 생성 : link(2)

```
#include <unistd.h>
int link(const char *existing, const char *new);
```

  - ✓ 두 경로는 같은 파일시스템에 존재해야 함
- 심볼릭 링크 생성 : symlink(2)

# 다수의 이름을 갖는 file(link)

- Hard link and link count (링크 계수)
  - A file with multiple names can have many link counts
  - This saves disk space and ensures many people can access the same file
  - `link` and `unlink` system calls
- The `link` system call

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

  - return:
    - success: 0
    - fail: -1
- How to move a file
  - Use `link` and `unlink` system calls
  - Use `rename` system call

# Program 예: move

```
/* move -- 한 화일을 하나의 경로이름으로부터 다른 경로이름으로 옮긴다. */
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

char *usage = "usage: move file1 file2\n";

/* main은 명령줄에 의해 표준적인 방법으로 전달된 인수를 사용한다. */
main (int argc, char **argv)
{
 if (argc != 3)
  fprintf (stderr, usage); exit (1);

 if (link (argv[1], argv[2]) == -1)
  perror ("link failed");exit (1);

 if (unlink (argv[1]) == -1)
  perror ("unlink failed"); unlink (argv[2]); exit (1);

 printf ("Succeeded\n");
 exit (0);
}
```

# Symbolic Link

- Hard link's limitations
  - ✓ directory link and link across file systems are not allowed
  - ✓ Symbolic link 는 그 자체가 하나의 file임(자신이 링크되어 있는 file에 대한 경로 수록)
  - ✓ Symbolic link에 의해 가리켜지고 있는 file 제거 시: link가 끊어짐.
- The **`symlink`** and **`readlink`** system calls

```
#include <unistd.h>
int symlink(const char *realname, const char *symname);
int readlink(const char *sympath, char *buffer, size_t bufsize);
```

  - ✓ return of `symlink`:
    - ➢ success: 0, fail: -1
  - ✓ `symname` 그 자체에 들어있는 데이터를 볼 경우 `readlink`를 사용
  - ✓ buffer: place to put the result pathname (not `NULL` terminating)
  - ✓ bufsize: the size of buffer
  - ✓ return of `readlink`:
    - ➢ success: # of characters, fail: -1

# 파일 링크

## 심볼릭 링크 생성 : symlink(2)

```
#include <unistd.h>
int symlink(const char *name1, const char *name2);
```

symlink 함수 사용하기

```
01  #include <sys/types.h>
02  #include <sys/stat.h>
03  #include <unistd.h>
04
05  int main(void) {
06      symlink("unix.txt", "unix.sym");
07
08      return 0;
09  }
```

```
# ls -l unix*
-rwxrwx---   2 root     other         24  1월   8일   15:47 unix.ln
-rwxrwx---   2 root     other         24  1월   8일   15:47 unix.txt
# ex3_9.out
# ls -l unix*
-rwxrwx---   2 root     other         24  1월   8일   15:47 unix.ln
lrwxrwxrwx   1 root     other          8  1월 11일   18:48 unix.sym ->
unix.txt
-rwxrwx---   2 root     other         24  1월   8일   15:47 unix.txt
```

# 심볼릭 링크 정보 검색

- 심볼릭 링크 파일 정보 검색 : lstat(2)

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

  - ✓ lstat : 심볼릭 링크 자체의 파일 정보 검색
  - ✓ 심볼릭 링크를 stat 함수로 검색하면 원본 파일에 대한 정보가 검색

- 심볼릭 링크의 내용 읽기 : readlink(2)

```
#include <unistd.h>
ssize_t readlink(const char *restrict path, char  *restrict buf, size_t bufsiz);
```

  - ✓ 심볼릭 링크의 데이터 블록에 저장된 내용 읽기
  - ✓ 원본파일의 경로 등

- 원본 파일의 경로 읽기 : realpath(3)

```
#include <stdlib.h>
char *realpath(const char *restrict  file_name,
char *restrict resolved_name);
```

  - ✓ 심볼릭 링크가 가리키는 원본 파일의 실제 경로명 출력

# lstat 함수 사용하기

```
01  #include <sys/types.h>
02  #include <sys/stat.h>
03  #include <unistd.h>
04  #include <stdio.h>
05
06  int main(void) {
07      struct stat buf;
08
09      printf("1. stat : unix.txt ---\n");
10      stat("unix.txt", &buf);
11      printf("unix.txt : Link Count = %d\n", (int)buf.st_nlink);
12      printf("unix.txt : Inode = %d\n", (int)buf.st_ino);
13
14      printf("2. stat : unix.sym ---\n");
15      stat("unix.sym", &buf);
16      printf("unix.sym : Link Count = %d\n", (int)buf.st_nlink);
17      printf("unix.sym : Inode = %d\n", (int)buf.st_ino);
18
19     printf("3. lstat : unix.sym ---\n");
20     lstat("unix.sym", &buf);
21      printf("unix.sym : Link Count = %d\n", (int)buf.st_nlink);
22      printf("unix.sym : Inode = %d\n", (int)buf.st_ino);
23
24      return 0;
25  }
```

# lstat 함수 사용하기

```
# ls -li unix*
192 -rwxrwx---   2 root     other     24  1월   8일   15:47 unix.ln
202 lrwxrwxrwx   1 root     other      8  1월 11일   18:48 unix.sym->unix.txt
192 -rwxrwx---   2 root     other     24  1월   8일   15:47 unix.txt
# ex3_10.out
1. stat : unix.txt ---
unix.txt : Link Count = 2
unix.txt : Inode = 192
2. stat : unix.sym ---
unix.sym : Link Count = 2
unix.sym : Inode = 192
3. lstat : unix.sym ---
unix.sym : Link Count = 1
unix.sym : Inode = 202
```

# readlink 함수 사용하기

```
01  #include <sys/stat.h>
02  #include <unistd.h>
03  #include <stdlib.h>
04  #include <stdio.h>
05
06  int main(void) {
07      char buf[BUFSIZ];
08      int n;
09
10      n = readlink("unix.sym", buf, BUFSIZ);
11      if (n == -1) {
12          perror("readlink");
13          exit(1);
14      }
15
16      buf[n] = '\0';
17      printf("unix.sym : READLINK = %s\n", buf);
18
19      return 0;
20  }
```

```
# ex3_11.out
unix.sym : READLINK = unix.txt
# ls -l unix.sym
lrwxrwxrwx   1 root other 8  1월  11일    18:48 unix.sym ->unix.txt
```

# realpath 함수 사용하기

```
01  #include <sys/stat.h>
02  #include <stdlib.h>
03  #include <stdio.h>
04
05  int main(void) {
06      char buf[BUFSIZ];
07
08      realpath("unix.sym", buf);
09      printf("unix.sym : REALPATH = %s\n", buf);
10
11      return 0;
12  }
```

```
# ex3_12.out
unix.sym : REALPATH = /export/home/jw/syspro/ch3/unix.txt
```

# rename System Call

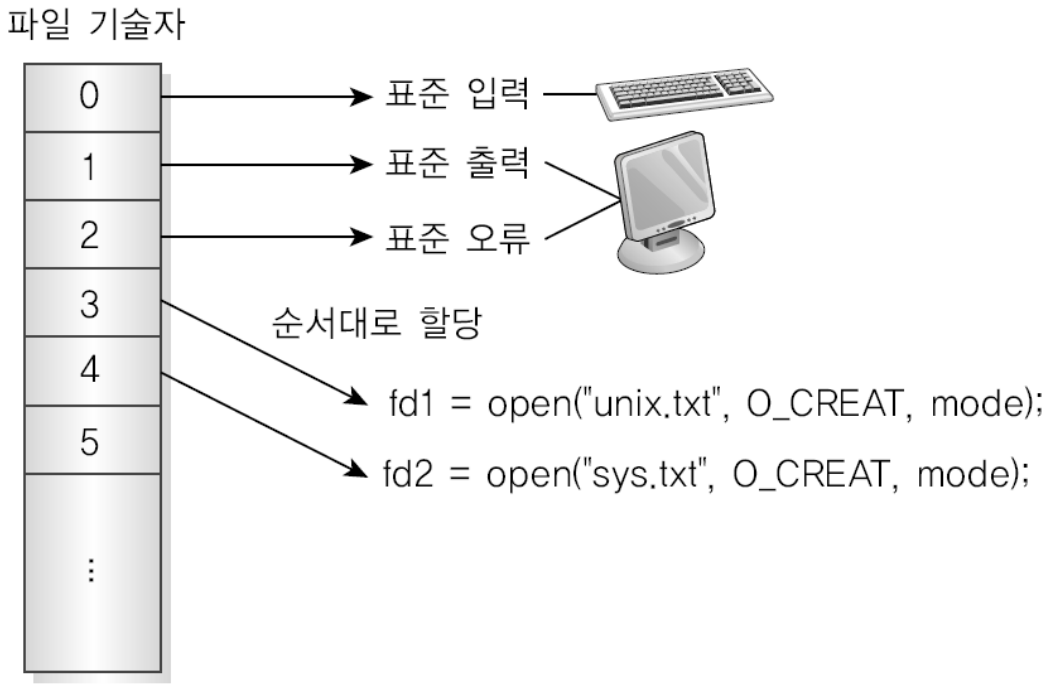
```
include <stdio.h>
int rename(const char *oldpathname, const char *newpathname);
```

- return:
  - ✓ success: 0
  - ✓ fail: -1

- regular file과 directory 이름의 재지정

# 파일 기술자

- 파일 기술자
  - ✓ 현재 열려있는 파일을 구분하는 정수값
  - ✓ 저수준 파일 입출력에서 열린 파일을 참조하는데 사용
  - ✓ 0번 : 표준 입력, 1번 : 표준 출력, 2번 : 표준 오류



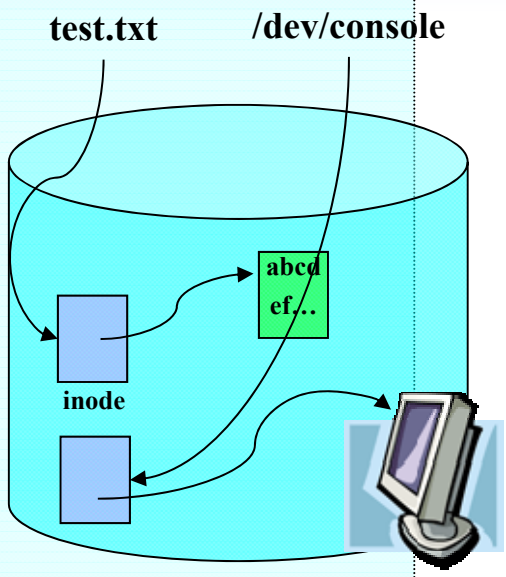

[그림 2-1] 파일 기술자 할당

# File Interface

- 파일 생성
  - creat(), open() with create option, mkfifo(), mknod()
  - inode와 데이터 블록들을 할당
- 파일 접근
  - open(), close(), read(), write()
  - inode와 task_struct 연결
- 파일 제어
  - stat()
  - lseek(), dup(), link()
  - mkdir(), readdir()
  - fcntl()
- 파일 시스템 제어
  - mount()
  - sync(), fsck()

# File Interface

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

#define MAX_BUF 4
char fname[] = "/usr/member/choijm/test.txt";
char tmp_data[] = "abcdefghijklmn";

int main()
{
        int fd, size;
        char buf[MAX_BUF];

        fd = open(fname, O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
        write(fd, tmp_data, sizeof(tmp_data));
        close(fd);

        fd = open(fname, O_RDONLY);
        lseek(fd, 5, SEEK_SET);
        size = read(fd, buf, MAX_BUF);
        close(fd);

        fd=open("/dev/console", O_WRONLY)
        write(fd, buf, MAX_BUF);
        close(fd);
}
```

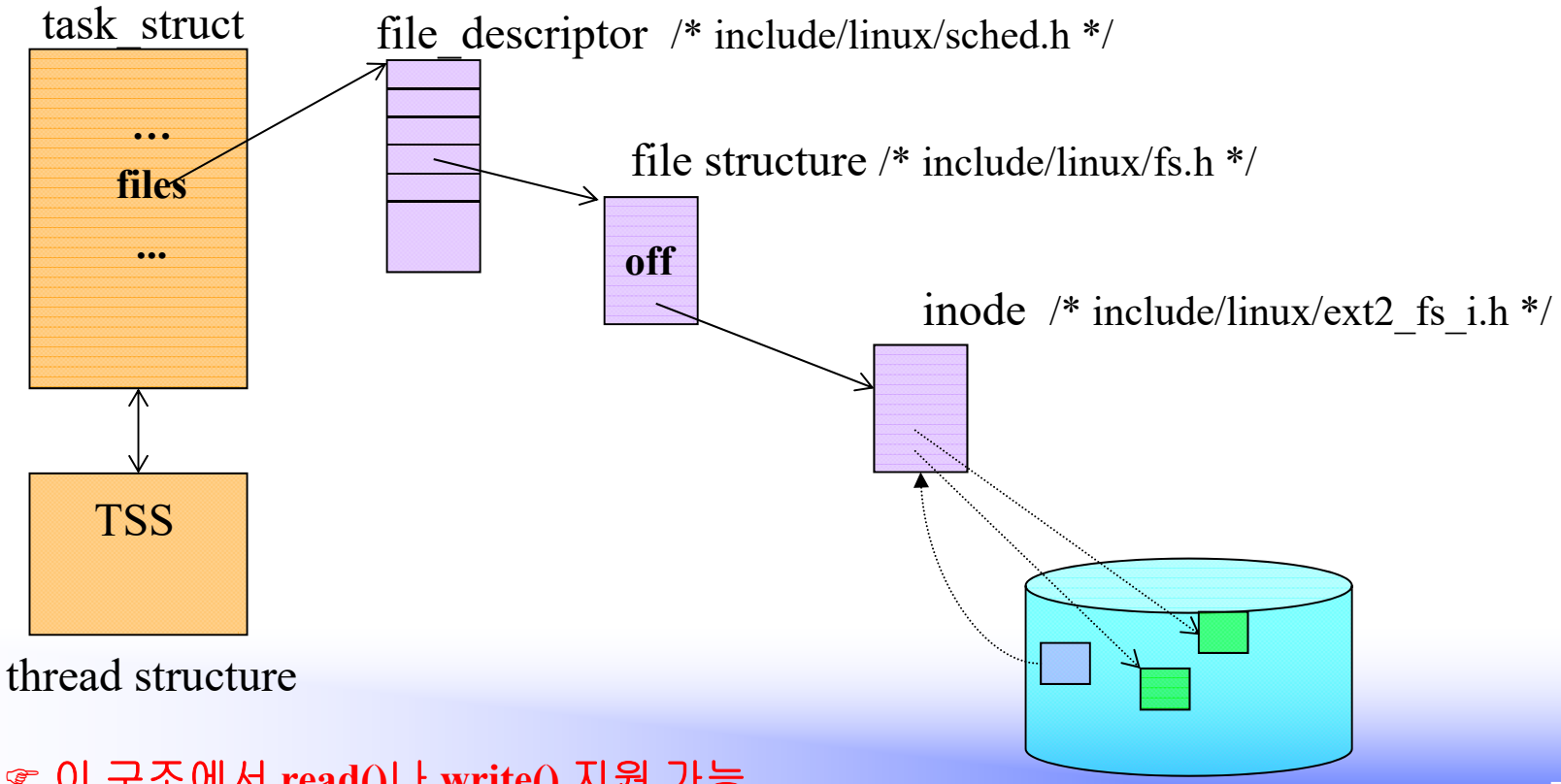



**offset**을 이동하는 인터페이스

터미널 같은 장치도 파일 인터페이스로 접근된다.

# File Interface

- fd (file descriptor): details of open() system call
  - ✓ 디스크에서 접근하려는 파일의 inode를 찾는다.
  - ✓ inode를 메모리로 읽는다.
  - ✓ inode와 task 자료 구조를 연결한다 (이때 fd 사용)




☞ 이 구조에서 **read()**나 **write()** 지원 가능

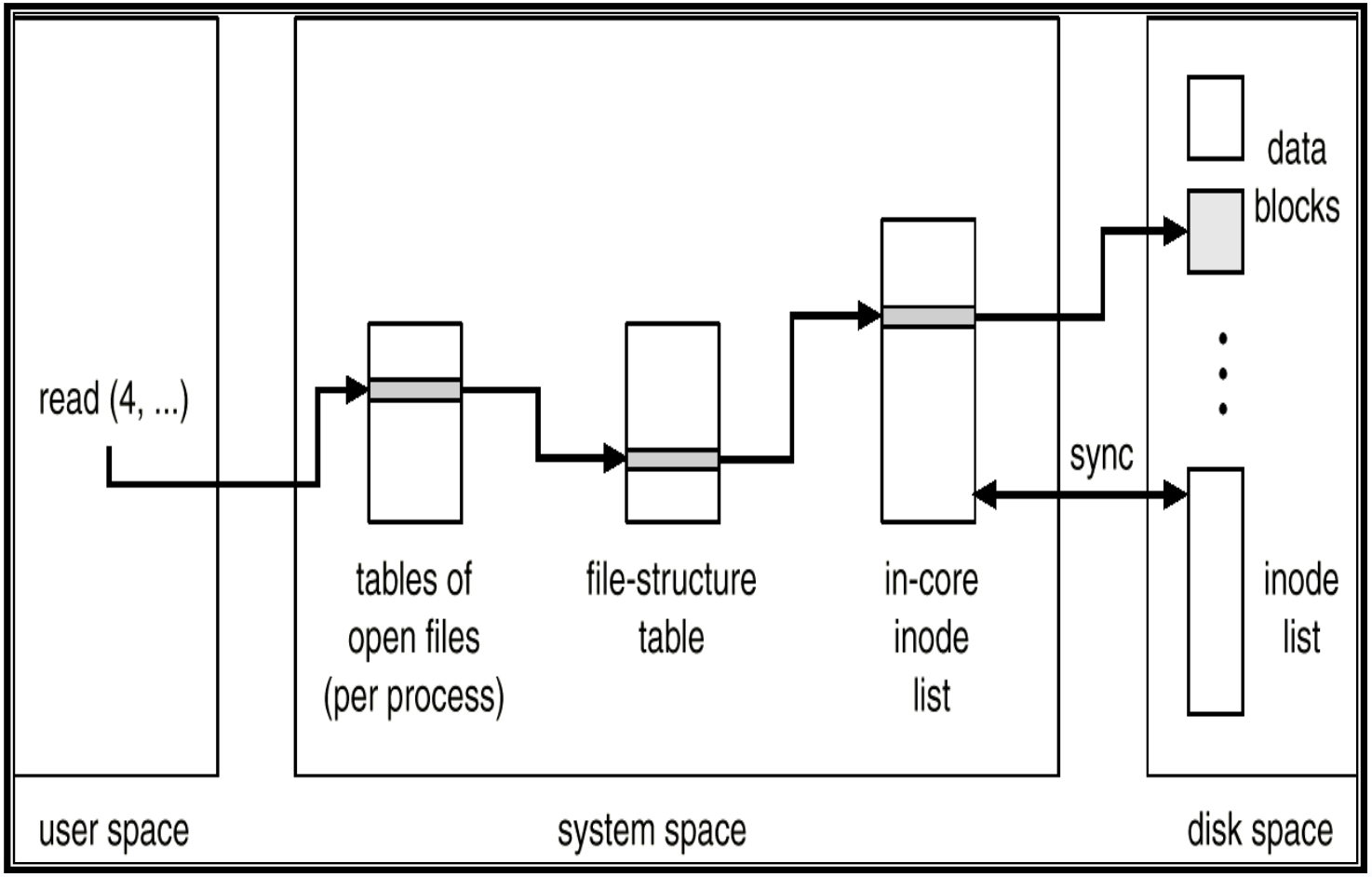
# Kernel data structures for open files

- user file descriptor table
  - ✓ allocated per process
  - ✓ identifies all open files for a process
  - ✓ when a process “open” or “creat” a file, the kernel allocates an entry
  - ✓ return value of “open” and “creat” is the index into the user file descriptor table
  - ✓ contains pointer to file table entry

# Kernel data structures for open files

- file table
  - ✓ global kernel structure
  - ✓ contains the description of all open files in the system
    - ➢ file status flag (open mode)
    - ➢ current file offset
  - ✓ contains pointer to in-core inode table entry
- in-core inode table
  - ✓ global kernel structure
  - ✓ when a process opens a file, the kernel converts the filename into an identity pair(device number, inode number)
  - ✓ the kernel then loads the corresponding inode into in-core inode table

# Kernel data structures for open files



# UNIX file access primitives

- `open`: opens a file to read, write or create a file
- `creat`: creates an empty file
- `close`: closes a previously opened file
- `read`: extracts information from a file
- `write`: places information into a file
- `lseek`: moves to a specified byte in a file
- `unlink`: removes a file
- `remove`: alternative method to remove a file
- `fcntl`: controls attributes associated with a file

# open() System Call

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *pathname, int flag [, mode_t mode]);
```

- `pathname`: absolute or relative path name
- `flag`: some macros in `<fcntl.h>`
  - ✓ `O_RDONLY`, `O_WRONLY`, `O_RDWR`, `O_CREAT`, `O_TRUNC`, `O_APPEND`
  - ✓ `O_BINARY`, `O_TEXT`
  - ✓ `mode: S_IWRITE, S_IREAD, S_IREAD | S_IWRITE`
- `mode`: used with `O_CREAT` flag
- Return of **open**:
  - ✓ success: a file descriptor(>=0), fail: -1

# open() System Call

```
/* 초보적인 프로그램 예 */

/* 이 헤더 파일들은 아래에서 논의한다 */
#include <fcntl.h>
#include <unistd.h>

main()
{
    int fd;
    ssize_t nread;
    char buf[1024];

    /* 화일 "data"를 읽기 위해 개방한다 */
     fd = open("data", O_RDONLY);

    /* 데이터를 읽어 들인다 */
    nread = read(fd, buf, 1024);

    /* 화일을 폐쇄한다 */
    close(fd);
}
```

# open() System Call

```
#include <stdlib.h>          /* exit 호출을 위한 것임 */
#include <fcntl.h>

char *workfile="junk";       /* workfile 이름을 정의한다 */

main()
{
 int filedes;

 /* <fcntl.h>에 정의된 O_RDWR을 사용하여 개방한다 */
 /* 화일을 읽기/쓰기로 개방한다 */

 if ((filedes = open (workfile, O_RDWR)) ==-1) {
        printf ("Couldn't open %s\n", workfile);
        exit (1);            /* 오류이므로 퇴장한다 */
 }

 /* 프로그램의 나머지 부분이 뒤따른다 */

 exit (0);                   /* 정상적인 퇴장 */
}
```

# open() System Call

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

#define PERMS 0644  /* O_CREAT를 사용하는 open을 위한 허가 */

char *filename="newfile";

main()
{
 int filedes;

 if ((filedes = open (filename, O_RDWR|O_CREAT, PERMS)) ==-1){
    printf ("Couldn't create %s\n",filename);
    exit (1);                /*오류이므로 퇴장한다*/
 }

 /* 프로그램의 나머지 부분이 뒤따른다 */

 exit (0) ;
}
```

# creat() System Call

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

- `pathname`: absolute or relative path name
- `mode`: access permission (in octal)
  - ✓ `4`(read), `2`(write) and `1`(execute) for owner, group and others
  - ✓ eg. `0644` means r/w for owner, r for group and others
- Return:
  - ✓ success: a file descriptor(>= 0)
  - ✓ fail: −1

```
filedes = creat("/tmp/newfile", 0644);
filedes = open("/tmp/newfile", O_WRONLY|O_CREAT|O_TRUNC, 0644);
```

# close() System Call

```
#include <unistd.h>
int close(int filedes);
```

- Return of close:
  - ✓ success: 0, fail: -1
- 프로그램의 수행이 끝나면 모든 개방된 파일은 자동적으로 close됨

```
filedes = open ("file", O_RDONLY);
.
.
.
close(filedes);
```

# read() System Call

```
#include <unistd.h>
ssize_t read(int fd, void *buffer, size_t n);
```

- `fd`: file descriptor returned from `open` or `creat`
- `buffer`: starting address where data is stored
  - ✓ The user program should reserve enough buffer area
- `n`: # of bytes to read
- Return:
  - ✓ # of successfully read bytes

```
int fd;
ssize_t nread;
char buffer[SOMEVALUE];

/* fd는 open에 대한 호출로부터 얻은 것임 */
.
.
.
nread = read(fd, buffer, SOMEVALUE);
```

# read–write

```
int fd;
ssize_t n1,n2;
char buf1[512], buf2[512];
. . .
if(( fd = open("foo", O_RDONLY)) ==-1)
   return (-1);

n1 = read(fd, buf1, 512);
n2 = read(fd, buf2, 512);
```

# read-write

```
/* count -- 한 파일내의 문자 수를 센다 */
#include <fcntl.h>
#define BUFSIZE 512

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

   /*"anotherfile"을 읽기 전용으로 개방 */
   if (( filedes = open ("anotherfile", O_RDONLY)) ==-1) {
      printf ("error in opening anotherfile\n");
      exit (1);
   }

   /* EOF까지 반복하라. EOF는 복귀값 0에 의해 표시된다. */
   while((nread = read(filedes, buffer, BUFSIZE)) >0)
      total += nread;          /* total을 증가시킨다. */

   printf ("total chars in anotherfile: %ld\n", total);
   exit (0);
}
```

# write() System Call

```
#include <unistd.h>
ssize_t write(int fd, void *buffer, size_t n);
```

- fd: file descriptor returned from open or creat
- buffer: 쓰여질 데이터에 대한 포인터
- n: # of bytes to write
- Return:
  - ✓ # of successfully written bytes

```
int fd;
ssize_t w1, w2;
char header1[512], header2[1024];
.
.
if((fd = open("newfile", O_WRONLY|O_CREAT|O_EXCL, 0644)) ==
  -1)
    return (-1);

w1 = write(fd, header1, 512);
w2 = write(fd, header2, 1024);
```

# The copyfile Example

```
#include <unistd.h>
#include <fcntl.h>
#define BUFSIZE 512
#define PERM 0644
int copyfile(const char *name1, const char *name2)
{
  int infile, outfile; ssize_t nread; char buffer[BUFSIZE];

  if((infile = open(name1, O_RDONLY))==-1) return -1;
  if((outfile = open(name2, O_WRONLY|O_CREAT|O_TRUNC, PERM))==-1){
     close(infile); return -2;
  }
  while((nread=read(infile, buffer, BUFSIZE))>0){
     if(write(outfile, buffer, nread)<nread){
        close(infile); close(outfile);
        return -3;
     }
  }
  close(infile); close(outfile);
  if(nread==-1) return -4;
  else return 0;
}
```

# read와 write의 효율성

- Results of `copyfile` test

| BUFSIZE | Real time | User time | System time |
|---|---|---|---|
| 1 | 24.49 | 3.13 | 21.16 |
| 64 | 0.46 | 0.12 | 0.33 |
| 512 | 0.12 | 0.02 | 0.08 |
| 4096 | 0.07 | 0.00 | 0.05 |
| 8192 | 0.07 | 0.01 | 0.05 |

  - ➔ 가장 좋은 성능은 시스템의 `blocking factor`의 배수 일 때

- 프로그램의 효율성 향상 방법
  - ✓ 시스템 호출의 횟수를 줄여야 한다

# dup() and dup2() System Calls

- synopsis

```
#include <unistd.h>
int dup(int oldfd);
int dup2(int oldfd, int newfd);
```

- description
  - ✓ create a copy of the file descriptor oldfd
  - ✓ close_on_exec flag is not copied.
  - ✓ The old and new descriptors may be used interchangeably
    - ➢ if the file position is modified by using lseek on one of the descriptors, the position is also changed for the other
    - ➢ 같은 개방 파일/디바이스
    - ➢ 같은 파일 포인터
    - ➢ 같은 액세스 모드

# dup() and dup2() System Calls

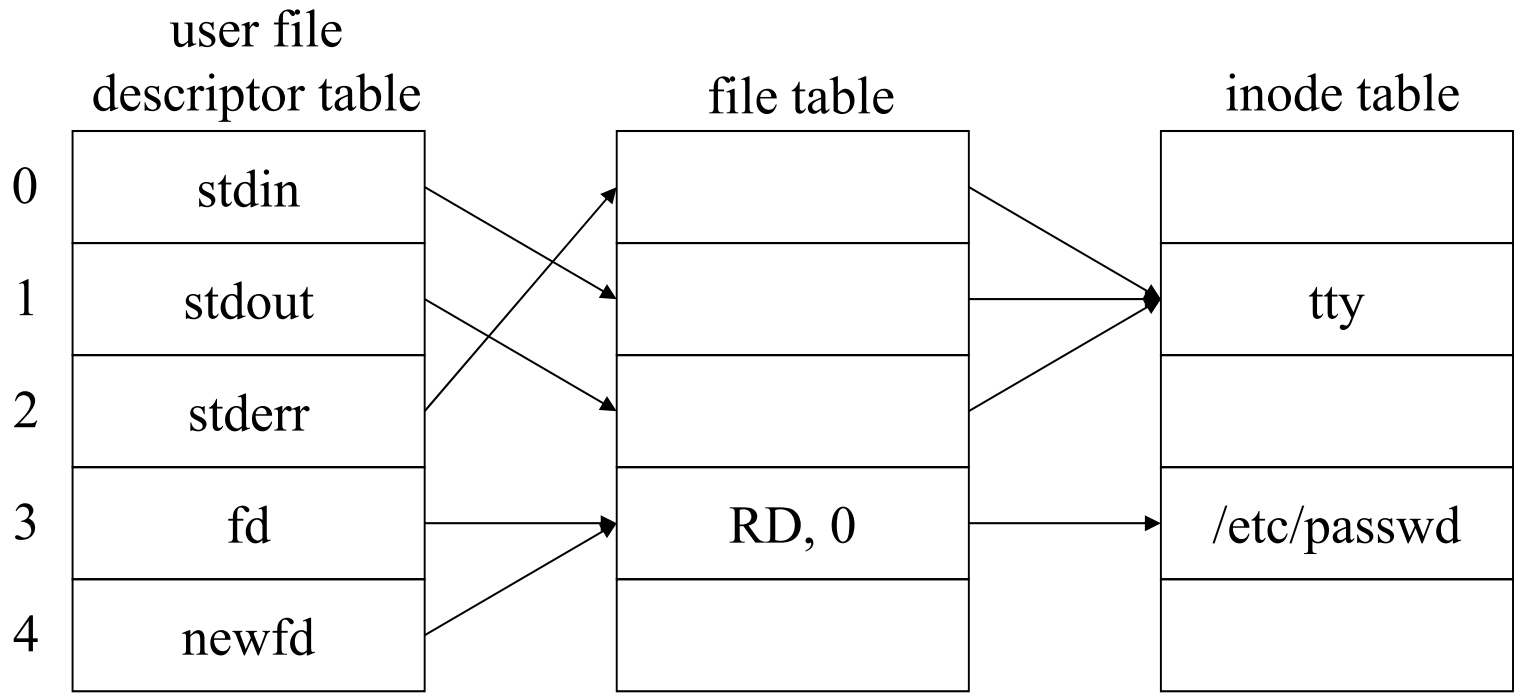
- ✓ two descriptors do not share the close-on-exec flag
- ✓ dup uses the lowest-numbered unused descriptor for the new descriptor
- ✓ dup2 makes newfd be the copy of oldfd, closing newfd first if necessary.
- ✓ return value
  - ➢ the new descriptor, or -1 if an error occurred

❑ example

1) int fd = dup(STDOUT_FILENO);
2) fd = open("input_file", O_RDONLY);
   dup2(fd, STDIN_FILENO);
3) fd = open("input_file", O_RDONLY);
   close(STDIN_FILENO);
   dup(fd);

# dup() and dup2() System Calls

```
fd = open("/etc/passwd", O_RDONLY);
newfd = dup(fd);
```



# dup() and dup2() System Calls

- Example

```
#include <unistd.h>
#include <fcntl.h>

int main(void)
{
        int fd;
        fd = creat("dup_result", 0644);
        dup2(fd, STDOUT_FILENO);
        close(fd);
        printf("hello world\n");
        return 0;
}
```

# dup() and dup2() System Calls

```
#include <fcntl.h>
#include <string.h>
#include <unistd.h>

main()
{
        char *fname = "test.txt";
        int fd1, fd2, cnt;
        char buf[30];

        fd1 = open(fname, O_RDONLY);
        if(fd1 < 0) {
                perror(”open( )");
                exit (-1);
        }
        fd2 = dup(fd1);
        cnt = read(fd1, buf, 12);
        buf[cnt] = '\0';
        printf("fd1's printf : %s\n", buf);
        lseek(fd1, 1, SEEK_CUR);
        cnt = read(fd2, buf, 12);
        buf[cnt] = '\0';
        printf("fd2's printf : %s\n", buf);
}
```

# dup() and dup2() System Calls

[test.txt의 내용]
Hello, Unix! How are you?

[수행 결과]

fd1's printf : Hello, Unix!
fd2's printf : How are you?

# dup() and dup2() System Calls

```
#include <fcntl.h>
main()
{
        char *fname = "test.txt";
        int  fd;

        if((fd = creat(fname, 0666)) < 0) {
                perror("creat( )");
                exit(-1);
        }
        printf("First printf is on the screen.\n");
        dup2(fd,1);
        printf("Second printf is in this file.\n");
}
```

[수행 결과]

```
$ a.out
First printf is on the screen.
$ cat test.txt
Second printf is in this file.
```

# lseek와 random access

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

- read-write 포인터의 위치 변경
- `fd`: file descriptor returned from **open** or **creat**
- `offset`: relative offset (지정위치로부터 바이트 수)
- `start_flag`:
  - ✓ **SEEK_SET:** 파일의 선두
  - ✓ **SEEK_CUR:** 현재 파일 포인터의 위치
  - ✓ **SEEK_END:** 파일의 끝
- Return:
  - ✓ success: a new position in the file
  - ✓ fail: -1
- lseek 는 fd로 접근, fseek는 fp로 접근

# lseek와 random access

```
off_t newpos;
.
.
newpos = lseek(fd, (off_t) -16, SEEK_END);
```

```
filedes = open(filename, O_RDWR);
lseek(filedes, (off_t)0, SEEK_END);
write(filedes, outbuf, OBSIZE);
```

```
off_t filesize;
int filedes;
.
.
filesize = lseek(filedes, (off_t)0, SEEK_END);
```

# The Hotel Example(1)

- residents: 호텔에 투숙한 사람들의 이름을 기록한 파일
- 각 줄의 길이는 41개의 문자 (41번째는 '\n')

```
/* getoccupier -- residents 화일로부터 투숙객의 이름을 얻는다. */
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#define NAMELENGTH 41

char namebuf[NAMELENGTH]; /*이름을 보관할 버퍼 */
int infile = -1;          /*화일 기술자를 보관할 것임 */

char *getoccupier(int roomno)
{
  off_t offset;
  ssize_t nread;

  /* 화일을 처음으로 개방한다 */
  if(infile==-1 && (infile=open("residents",O_RDONLY)) == -1){
    return (NULL);        /* 화일을 개방하지 못함 */
  }
  offset = (roomno - 1) * NAMELENGTH;
```

# The Hotel Example(2)

```
 /* 방의 위치를 찾아 투숙객의 이름을 읽는다 */
 if (lseek(infile, offset, SEEK_SET) == -1) return (NULL);
 if ((nread = read (infile, namebuf, NAMELENGTH)) <= 0) return (NULL);
 /* 개행문자를 널 종결자(terminator)로 대체하여 하나의 스트링을 생성하라. */
 namebuf[nread -1} = '\0';
 return (namebuf);
}

/* list.c -- 모든 투숙객의 이름을 리스트하라 */
#define NROOMS        10
int main()
{
 int j;
 char *getoccupier (int), *p;
 for ( j = 1; j<= NROOMS; j++) {
  if (p = getoccupier(j))
       printf ("Room %2d, %s\n", j, p);
  else
       printf ("Error on room %d\n", j);
 }
}
```

# 한 파일 끝에 자료 추가 하기

```
/* 화일의 끝으로 이동 */
lseek(filedes, (off_t)0, SEEK_END);
write(filedes, appbuf, BUFSIZE);
```

- 보다 일반적인 방법

```
filedes = open("yetanother", O_WRONLY | O_APPEND);
write(filedes, appbuf, BUFSIZE);
```

# 파일 속성 제어 – fcntl () System Call

- 파일에 대한 속성을 조절하는 함수

```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
int fcntl(int fd, int cmd, ...);
```

- fd: file descriptor returned from open or creat
- cmd: F_GETFL, F_SETFL, …
- Return:
  - ✓ success: an integer(>=0)
    - ➢ compute Return & O_ACCMODE
    - ➢ can be one of O_RDONLY, O_WRONLY, …
  - ✓ fail: -1

# fcntl () System Call 매개변수 – cmd

- F_DUPFD: 열려진 파일 지정자를 복사.
  - ✓ 언뜻 보면 dup2(2) 함수와 매우 비슷한데, dup2 는 복사될 파일지정자를 사용자가 지시하는 반면, F_DUPFD 를 사용할 경우 arg 와 같은 크기의 파일지정자를 되돌려주거나, 이미 사용되어지고 있다면, arg 보다 큰 할당 가능한 파일지정번호 중 가장 작은 번호를 되돌려준다.
  - ✓ 복사된 파일지정번호는 잠금, 파일위치 포인터, 플레그 등을 공유한다. 즉 파일지정자들 중 하나에서 파일의 위치가 변경된다면(lseek등을 이용), 다른 파일지정자도 변경된다.
  - ✓ 그렇지만 close-on-exec 는 공유하지 않는다.
- F_GETFD: 리턴값으로 fd 에 대한 flag값을 전달. 현재는 FD_CLOEXEC 정보만 전달
  - ✓ FD_CLOEXEC 는 close-on-exec 정책에 관한 내용
- F_SETFDFD_CLOEXEC(close-on-exec) 의 값을 지정된 비트값으로 설정한다.
- F_GETFL: 파일지정자에 대한 플래그값 – open(2) 호출 시 지정한 플래그를 되돌려준다.
- F_SETFL: arg 에 지정된 값으로 파일지정자 fd 의 플래그를 재 설정
  - ✓ 현재는 단지 O_APPEND, O_NONBLOCK, O_ASYNC 만을 설정할 수 있다. 다른 플래그들 (O_WRONLY 와 같은)은 영향을 받지 않는다.
- F_GETOWN:
  - ✓ 비동기 입출력과 관련되어서 사용
  - ✓ SIGIO와 SIGURG 신호를 받는 프로세스 아이디를 얻기 위해서 사용
- F_SETOWN:
  - ✓ 비동기 입출력과 관련되어서 사용
  - ✓ SIGIO, SIGURG 시그널을 수신하는 프로세스 아이디(혹은 그룹)을 설정하기 위해서 사용
- F_SETLK, F_SETLKW, F_SETLK – 레코드 잠금에 대한 설정도 가능

# fcntl System Call

```
/* filestatus -- 화일의 현재 상태를 기술한다. */

#include <fcntl.h>

int filestatus(int filedes)
{
 int arg1;

 if(( arg1 = fcntl (filedes, F_GETFL)) == -1){
    printf ("filestatus failed\n");
    return (-1);
 }

 printf("File descriptor %d: ",filedes);
```

```
/* 개방시의 플래그를 테스트한다. */
 switch ( arg1 & O_ACCMODE){
 case O_WRONLY:
          printf ("write-only");
          break;
 case O_RDWR:
          printf ("read-write");
          break;
 case O_RDONLY:
          printf ("read-only");
          break;
 default:
          printf("No such mode");
 }

 if (arg1 & O_APPEND)
   printf (" -append flag set");

 printf ("\n");
 return (0);
}
```

# The unlink/remove System Call

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

#include <stdio.h>
int remove(const char *pathname);
```

- 파일의 제거
- pathname: absolute or relative path name
- Return:
  - ✓ success: 0
  - ✓ fail: -1
- remove
  - ✓ 디렉토리에 대해서는 rmdir( )을 호출
  - ✓ 파일에 대해서는 unlink ()를 호출

# [예제 2-10] unlink 함수로 파일 삭제하기

```
01  #include <unistd.h>
02  #include <stdlib.h>
03  #include <stdio.h>
04
05  int main(void) {
06      int cnt;
07
08      cnt = unlink("tmp.aaa");
09      if (cnt == -1) {
10          perror("Unlink tmp.aaa");
11          exit(1);
12      }
13
14      printf("Unlink tmp.aaa success!!!\n");
15
16      return 0;
17  }
```

tmp.aaa 파일 삭제

```
# ls -l tmp*
-rw-r--r--   1 root     other         37  1월  6일  17:50 tmp.aaa
-rw-r--r--   1 root     other         35  1월  6일  18:06 tmp.bbb
# ex2_10.out
Unlink tmp.aaa success!!!
# ls -l tmp*
-rw-r--r--   1 root     other         35  1월  6일  18:06 tmp.bbb
```

# Standard input, ouput and Error

- Standard input, output and error
  - ✓ standard input(fd=0): keyboard
  - ✓ standard output(fd=1): terminal screen
  - ✓ standard error(fd=2): terminal screen
- File redirections on UNIX shell
  - ✓ standard input redirection:

```
$ command < file
```

  - ✓ standard output redirection:

```
$ command > file
```

  - ✓ standard error redirection:

```
$ command 2> file
```

# I/O의 예

```
/* io -- 표준 입력을 표준 출력으로 복사 */

#include <stdlib.h>
#include <unistd.h>

#define SIZE 512

main()
{
  ssize_t nread;
  char buf[SIZE];

  while ((nread = read (0, buf, SIZE)) > 0)
      write (1, buf, nread);
  exit (0);
}
```

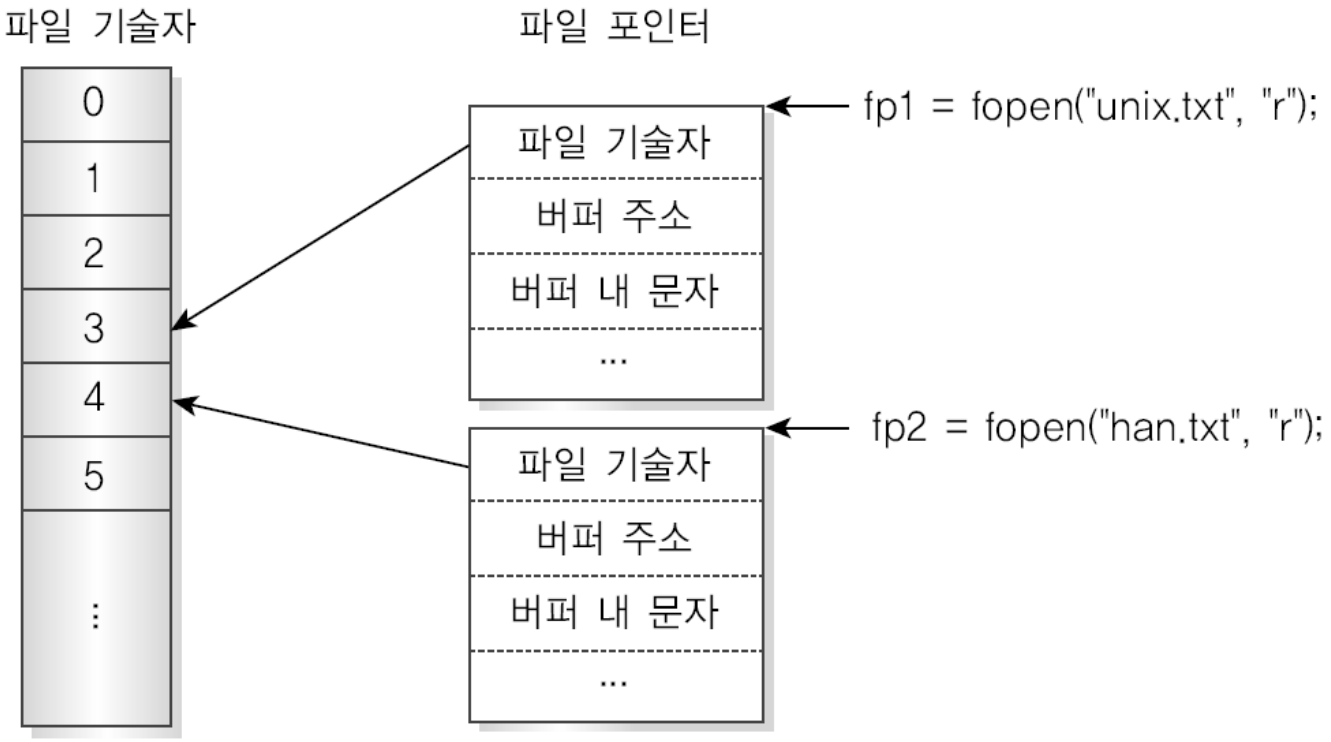
# The Standard I/O Library

- *file pointer*
  - ✓ 특정 파일 기술자를 통해 읽혀질/쓰여질 파일의 다음 바이트 위치를 기록 (bookmark)
  - ✓ 주로 file pointer를 이용하나 file handle을 이용 가능

- The standard I/O library
  - ✓ ANSI C standard
  - ✓ Offers many more facilities than the system calls
  - ✓ Offers efficient mechanisms
  - ✓ `FILE *` instead of file descriptor
  - ✓ eg. `fopen()`, `getchar()`, `putchar()`, `getc()`, `putc()`, `scanf()`, `printf()`, …

# 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);
```

- ✓ filename으로 지정한 파일을 mode로 지정한 모드에 따라 열고 파일 포인터를 리턴

| 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);
```

# 버퍼 기반 입출력

- 버퍼 기반 입력함수: fread(3)

```
#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

# fopen() Library

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

int main()
{
  FILE *stream;

  if (( stream = fopen ("junk", "r")) == NULL) {
     printf ("Could not open file junk\n");
     exit (1);
  }
}
```

# file pointer 예제

```
#include <stdio.h>
#define MAXHOST 16
struct record {
    int vdo;
    int fps;
    int odo;
};
typedef struct param_tag {
    int ndwin;
    int recflag;
    struct record rec[MAXHOST];
}Parameter;

void display_param (Parameter *param);
int fwrite_param (Parameter *param);
int fread_param (Parameter *param);

int main ()
{
   int i;
   Parameter param;
```

```
    if (!fread_param (&param)) {
        param.ndwin = MAXHOST;

        for (i=0; i<MAXHOST; i++)  {
            param.rec[i].vdo = i%2;
            param.rec[i].fps = 15;
            param.rec[i].odo = i%2 + 1;
        }
    } else {
        for (i=0; i<MAXHOST; i++) {
            param.rec[i].vdo = i;
            param.rec[i].fps = 30;
            param.rec[i].odo = i%5 + 1;
        }
    }

    display_param (&param);
    fwrite_param (&param);

    return 1;
}
```

```
/* Display a directory */
void display_param (Parameter *param)
{
   int i;
   fprintf (stderr, "num. of video display partition = %d₩n", param->ndwin);

   for (i=0; i<MAXHOST; i++)
      fprintf (stderr, "%4d: vdo=%2d : %2d fps, odo=%2d ₩n", i, param->rec[i].vdo, param->rec[i].fps,
      param->rec[i].odo);
}

/* Save the list to a disk file */
int fwrite_param (Parameter *param)
{
   FILE *fp;
   char fname[512];
   sprintf(fname, "%s/param.conf", getenv("PWD"));
   if (!(fp = fopen(fname, "w+"))) {
      fprintf(stderr, "Error: opening File to write.₩n");
      return 0;
   }
```

```
    fwrite (param, sizeof (Parameter), 1, fp);
    fclose (fp);
    return 1;
}

/* Load the list from disk file */
int fread_param (Parameter *param)
{
   FILE *fp;
   char fname[512];
   sprintf(fname, "%s/param.conf",
      getenv("PWD"));
   if ((fp = fopen (fname, "rb")) == NULL) {
      fprintf (stderr, "Error: can't open file to
      read.₩n");
      return 0;
   }
```

```
while (!feof (fp)) {
    if (fread (param, sizeof (Parameter), 1, fp) != 1) {
      fclose (fp);
      return 1;
    }
  }

  fclose (fp);
  return 1;
}
```

# fprintf를 이용한 오류 메시지 출력

```
#include <stdio.h>   /*표준 에러의 정의를 위해 */
...
fprintf(stderr, "error number %d\n", errno); //stderr은 FILE *
```

```
/* notfound  화일 오류를 출력하고 퇴장(exit)한다. */

#include <stdio.h>
#include <stdlib.h>
int notfound (const char *progname, const char *filename)
{
  fprintf (stderr,
              "%s: file %s not found\n", progname, filename);
  exit (1);
}
```

# 문자 기반 입출력 함수

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

```
#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.

- 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);
```

# 문자 기반 입출력 함수 사용하기

```
01  #include <stdlib.h>
02  #include <stdio.h>
03
04  int main(void) {
05      FILE *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)

```
#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);
```

# 문자열 기반 입출력 함수 사용하기

```
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
```

# getc(), putc() Library

```
#include <stdio.h>
int getc(FILE *istream);    /* istream으로부터 한 문자를 읽어라. */
int putc(int c, FILE *ostream);  /* ostream에 한 문자를 넣어라. */
```

- 예

```
int c;
FILE *istream, *ostream;

/* istream을 읽기전용으로 개방하고, ostream을 쓰기전용으로 개방하라. */
.
.
while(( c = getc (istream)) !=EOF)
    putc(c, ostream);
```

# 형식 기반 입출력

- 형식 기반 입력 함수: scanf(3), fscanf(3)

```
#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.

# 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
```

# 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)

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

  - ✓ 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);
```

  - ✓ 현재 오프셋을 리턴. 오프셋은 파일의 시작에서 현재 위치까지의 바이트 수

# 파일 포인터의 위치 이동 – fseek()

int fseek (FILE *fp, long int offset, int whence);

long int offset: 새로운 위치까지의 변위차

int whence: 오프셋 시작점

SEEK_SET : 파일의 선두

SEEK_CUR : 현재 위치

SEEK_END : 파일의 맨 끝

- ✓ 파일 포인터의 위치를 whence 로 지정한 위치로부터 offset 바이트 만큼 이동
- ✓ 텍스트 모드 파일인 경우 offset은 0이거나 **ftell() 반환 값**

❑ long int ftell (FILE *fp);

- ✓ 파일의 선두로부터 현재 파일 포인터가 있는 곳까지 바이트 수 반환

# 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)

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

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

# 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
15      cur = ftell(fp);
16      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

# 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
```

# fseek ()

```
#include <stdio.h>

long filesize (FILE *stream)
{
     long curpos, length;

     curpos = ftell (stream); // 파일의 맨처음에 위치
     fseek (stream, 0L, SEEK_END); // fp를 파일의 맨뒤로 이동
     length = ftell (stream); // 파일의 맨끝과 처음 사이의 바이트 수
     fseek (stream, curpos, SEEK_SET); // fp를 파일의 맨 처음으로 위치
     return length;
}

int main ()
{
    FILE *stream;
    stream = fopen ("myfile.txt", "r");

     printf ("file size of  the file: %ld Bytes ₩n",  filesize (stream));
    return 0;
}
```

# 프로그래밍 실습:

- 명령행 인자를 사용하여 파일의 크기를 알아내는 프로그램을 작성하시오.
  - ✓ 명령행 인자에 대상 파일 이름을 대입

# 파일기술자와 파일포인터간 변환

- 저수준 파일 입출력의 파일기술자와 고수준 파일 입출력의 파일포인터는 상호 변환 가능

| 기능 | 함수원형 |
|---|---|
| 파일 기술자 → 파일 포인터 | FILE *fdopen(int fildes, const char *mode); |
| 파일 포인터 → 파일 기술자 | int fileno(FILE *stream); |

- 파일 포인터 생성: fdopen(3)

```
#include <stdio.h>
FILE *fdopen(int fildes, const char *mode);
```

  - ✓ 파일 기술자와 모드값을 받아 파일 포인터를 생성

- 파일 기술자 생성: fileno(3)

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

  - ✓ 파일 포인터를 인자로 받아 파일 기술자를 리턴

# fdopen 함수 사용하기

```
01  #include <fcntl.h>
02  #include <stdlib.h>
03  #include <stdio.h>
04
05  int main(void) {
06      FILE *fp;
07      int fd;
08      char str[BUFSIZ];
09
10      fd = open("unix.txt", O_RDONLY);
11      if (fd == -1) {
12          perror("open");
13          exit(1);
14      }
15
16      fp = fdopen(fd, "r");
17
18      fgets(str, BUFSIZ, fp);
19      printf("Read : %s\n", str);
20
21      fclose(fp);
22
23      return 0;
24  }
```

저수준 파일입출력 함수로 파일 오픈

파일 포인터 생성

고수준 파일읽기 함수로 읽기

```
# ex2_18.out
Read : Unix System Programming
```

# fileno 함수 사용하기

```
01  #include <unistd.h>
02  #include <fcntl.h>
03  #include <stdlib.h>
04  #include <stdio.h>
05
06  int main(void) {
07      FILE *fp;
08      int fd, n;
09      char str[BUFSIZ];
10
11      fp = fopen("unix.txt", "r");
12      if (fp == NULL) {
13          perror("fopen");
14          exit(1);
15      }
16
17      fd = fileno(fp);
18      printf("fd : %d\n", fd);
19
20      n = read(fd, str, BUFSIZ);
21      str[n] = '\0';
22      printf("Read : %s\n", str);22
23
24      close(fd);
25
26      return 0;
27  }
```

고수준 파일입출력 함수로 파일 오픈

파일 기술자 리턴

저수준 파일읽기 함수로 읽기

```
# ex2_19.out
fd : 3
Read : Unix System Programming
```

# 임시 파일 사용[1]

- 임시파일명이 중복되지 않도록 임시파일명 생성
- 임시파일명 생성: tmpnam(3)

```
#include <stdio.h>
char *tmpnam(char *s);
```

  - ✓ 임시 파일명을 시스템이 알아서 생성

- 접두어 지정: tempnam(3)

```
#include <stdio.h>
char *tempnam(const char *dir, const char *pfx);
```

  - ✓ 임시파일명에 사용할 디렉토리와 접두어 지정, 접두어는 5글자까지만 지원

```
char *fname;
fname = tempnam("/tmp", "hanbit");
```

# 임시 파일 사용[2]

- 템플릿을 지정한 임시 파일명 생성: mktemp(3)

```
#include <stdlib.h>
char *mktemp(char *template);
```

  - ✓ 임시파일의 템플릿을 받아 임시 파일명 생성
  - ✓ 템플릿은 대문자 'X'6개로 마치도록 해야한다.

```
/tmp/hanbitXXXXXX
```

# 임시 파일명 만들기

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  #include <string.h>
04
05  int main(void) {
06      char *fname;
07      char fntmp[BUFSIZ];
08      char template[32];
09
10      fname = tmpnam(NULL);
11      printf("1. TMP File Name(tmpnam) : %s\n", fname);
12
13      tmpnam(fntmp);
14      printf("2. TMP File Name(tmpnam) : %s\n", fntmp);
15
16      fname = tempnam("/tmp", "hanbit");
17      printf("3. TMP File Name(tempnam) : %s\n", fname);
18
19      strcpy(template, "/tmp/hanbitXXXXXX");
20      fname = mktemp(template);
21      printf("4. TMP File Name(mktemp) : %s\n", fname);
22
23      return 0;
24  }
```

```
# ex2_20.out
1. TMP File Name(tmpnam) : /var/tmp/aaaFUaGOe
2. TMP File Name(tmpnam) : /var/tmp/baaGUaGOe
3. TMP File Name(tempnam) : /tmp/hanbiAAAHUaGOe
4. TMP File Name(mktemp) : /tmp/hanbitIUaGOe
```

# 임시 파일의 파일 포인터 생성

- tmpfile(3)
  - ✓ 자동으로 w+ 모드로 열린 파일 포인터를 리턴

```
#include <stdio.h>
FILE *tmpfile();
```

tmpfile 함수 사용하기

```
01  #include <stdio.h>
02
03  int main(void) {
04      FILE *fp;
05
06      fp = tmpfile();
07
08      fputs("unix system", fp);
09
10      fclose(fp);
11
12      return 0;
13  }
```

임시 파일에 출력

# 프로그래밍 실습:

- 강의자료실 실습 코드 중에서 제 4 장 프로그래밍 코드(fparser)를 사용하여 초기화 파일의 내용을 다음과 같이 작성하여 화면에 출력하시오.
  - ✓ NAME, COUNTRY, AREA, PHONE, EMAIL, USRID, USIM
  - ✓ 분리자는 '=' 로 한다.

# 요약

## ❑ 파일

- ✓ 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.

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