Lecture Note 3.
File Programming

September 24, 2025

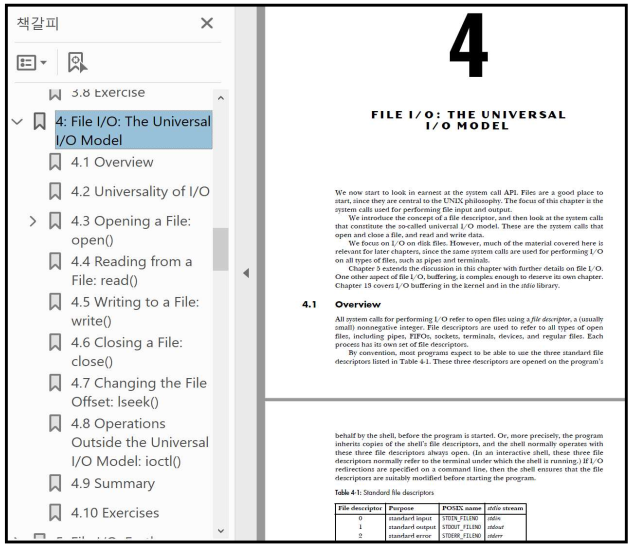
Jongmoo Choi
Dept. of Software
Dankook University
http://embedded.dankook.ac.kr/~choijm

(본 교재는 2025년도 과학기술정보통신부 및 정보통신기획평가원의 'SW중심대학사업' 지원을 받아 제작 되었습니다.)

Objectives

- Understand disk geometry
- Discuss system programs for disk (and storage)
- Apprehend the internal structure of a file
- Learn how to use file-related system calls
- Make a program (command) that manipulates a file

- Refer to Chapter 3, 4, 5 in the LPI and Chapter 10 in the CSAPP



책갈피

- 3.8 Exercise
- 4: File I/O: The Universal I/O Model
  - 4.1 Overview
  - 4.2 Universality of I/O
  - 4.3 Opening a File: open()
  - 4.4 Reading from a File: read()
  - 4.5 Writing to a File: write()
  - 4.6 Closing a File: close()
  - 4.7 Changing the File Offset: lseek()
  - 4.8 Operations Outside the Universal I/O Model: ioctl()
  - 4.9 Summary
  - 4.10 Exercises

4

FILE I/O: THE UNIVERSAL I/O MODEL

We now start to look in earnest at the system call API. Files are a good place to start, since they are central to the UNIX philosophy. The focus of this chapter is the system calls used for performing file input and output.

We introduce the concept of a file descriptor, and then look at the system calls that constitute the so-called universal I/O model. These are the system calls that open and close a file, and read and write data.

We focus on I/O on disk files. However, much of the material covered here is relevant for later chapters, since the same system calls are used for performing I/O on all types of files, such as pipes and terminals.

Chapter 5 extends the discussion in this chapter with further details on file I/O. One other aspect of file I/O, buffering, is complex enough to deserve its own chapter. Chapter 13 covers I/O buffering in the kernel and in the *stdio* library.

4.1 **Overview**

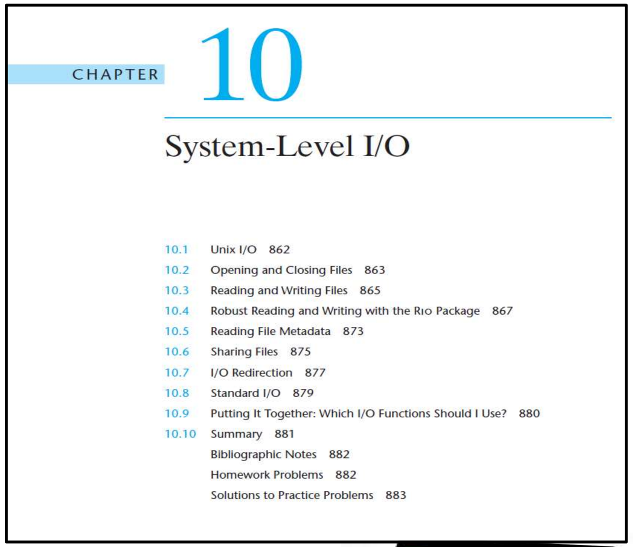
All system calls for performing I/O refer to open files using a *file descriptor*, a (usually small) nonnegative integer. File descriptors are used to refer to all types of open files, including pipes, FIFOs, sockets, terminals, devices, and regular files. Each process has its own set of file descriptors.

By convention, most programs expect to be able to use the three standard file descriptors listed in Table 4-1. These three descriptors are opened on the program's

behalf by the shell, before the program is started. Or, more precisely, the program inherits copies of the shell's file descriptors, and the shell normally operates with these three file descriptors always open. (In an interactive shell, these three file descriptors normally refer to the terminal under which the shell is running.) If I/O redirections are specified on a command line, then the shell ensures that the file descriptors are suitably modified before starting the program.

Table 4-1: Standard file descriptors

File descriptor	Purpose	POSIX name	*stdio* stream
0	standard input	STDIN_FILENO	*stdin*
1	standard output	STDOUT_FILENO	*stdout*
2	standard error	STDERR_FILENO	*stderr*



CHAPTER 10

System-Level I/O

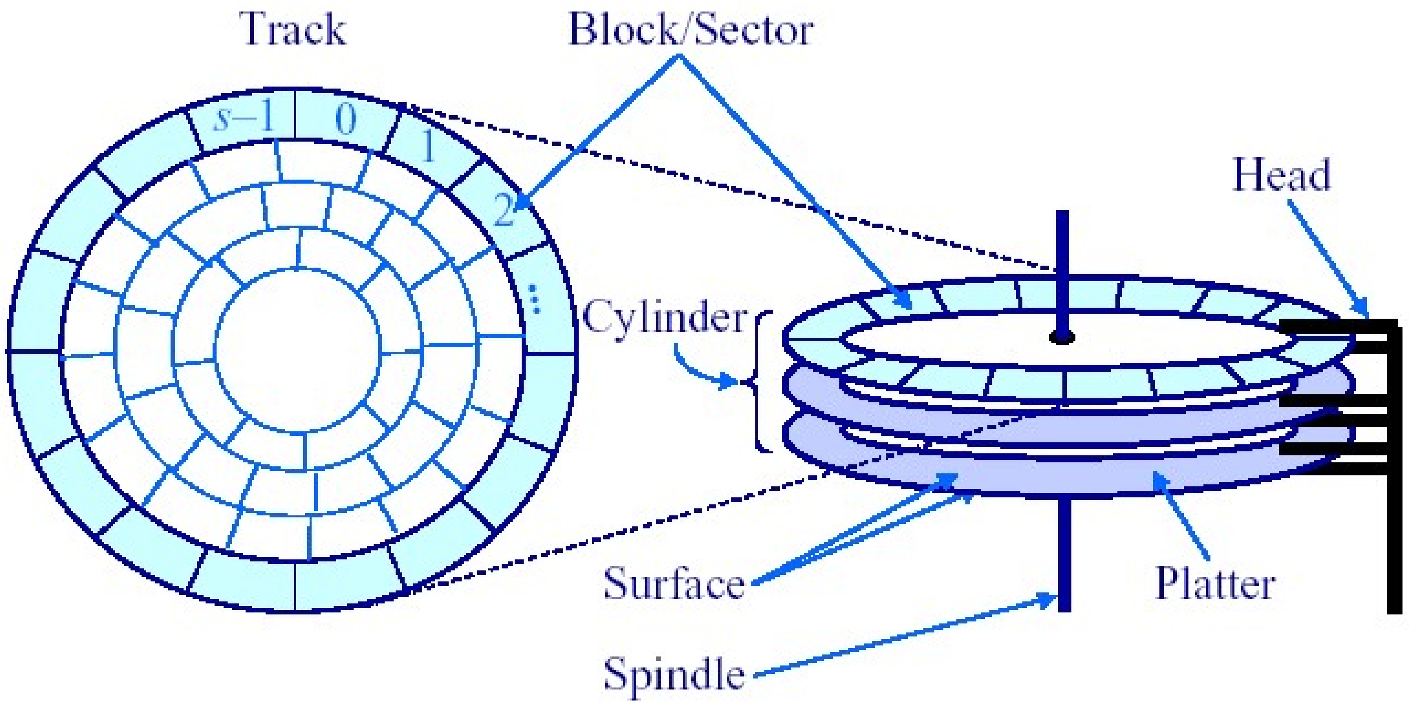
Introduction

- Issues on file
  - ✓ File manipulation (create, access, remove, …)
  - ✓ Manage file attributes/access control
  - ✓ Associate a file name with actual data stored in disk (regular file)
  - ✓ Support hierarchy structure (directory)
  - ✓ Support a variety of file types (device file, pipe, socket, …)

- File related system calls
  - ✓ open(), creat(): create a file, start accessing a file (authentication)
  - ✓ read(), write(): read/write bytes from/to a file
  - ✓ close(): finish accessing a file
  - ✓ lseek(): jump to a particular offset (location) in a file
  - ✓ unlink(), remove(): delete a file
  - ✓ stat(), fstat(): return information about a file
  - ✓ …

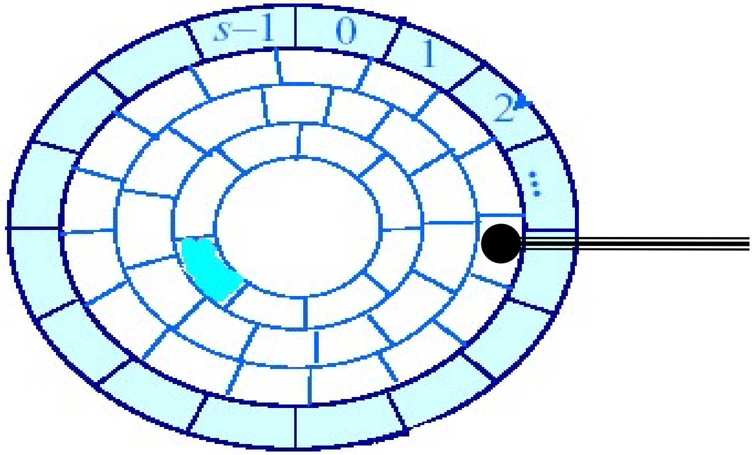
Disk structure (1/4)

- Components
  - ✓ Platter, Spindle, Surface
  - ✓ Track, Sector, Cylinder
  - ✓ Head, ARM



Disk structure (2/4)

- Disk access
  - ✓ Addressing
    - LBA (Logical Block Address)
    - head(surface), track(cylinder), sector
  - ✓ Access time
    - Seek time: move head to appropriate track
    - Rotational latency: wait for the sector to appear under the head
    - Transmission time: read/write the request sector(s)



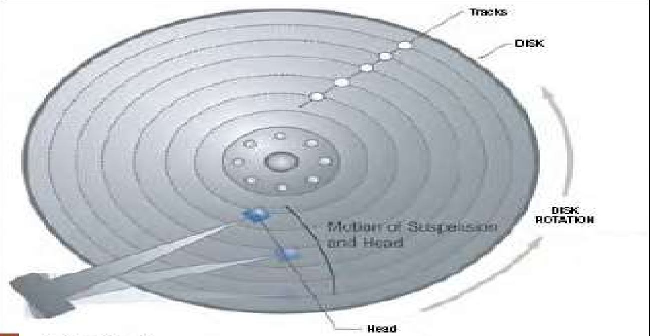


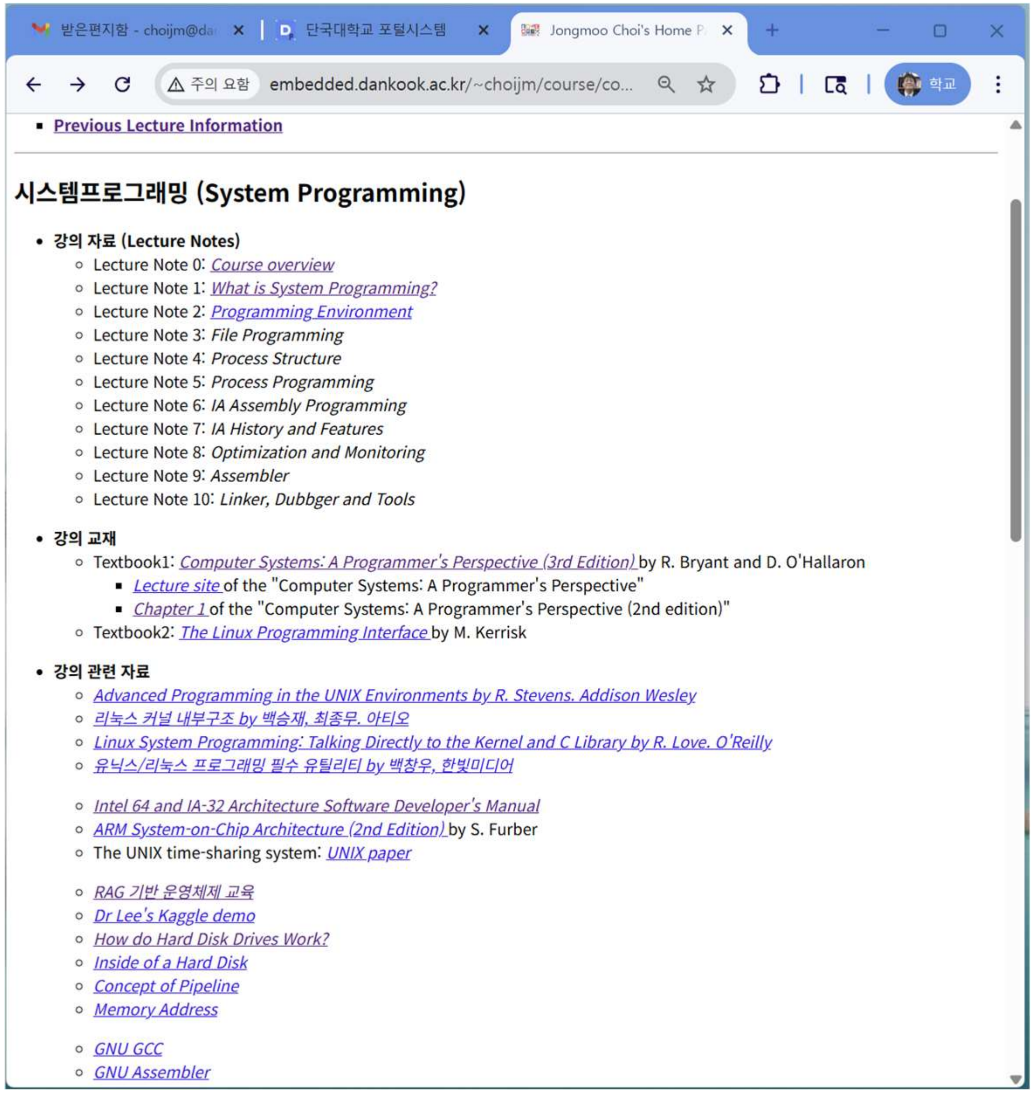
Figure 2

In order to read or write data on the disk, the read/write head needs to be physically moved to the correct place on the disk. That's a key reason why HDDs suffer from slow access time.

  - ✓ Try to reduce the Seek time and Rotational latency
    ➔ Make use of various disk scheduling (eg. SCAN or elevator algorithm) and Parallel access techniques (RAID)

Disk structure (3/4)

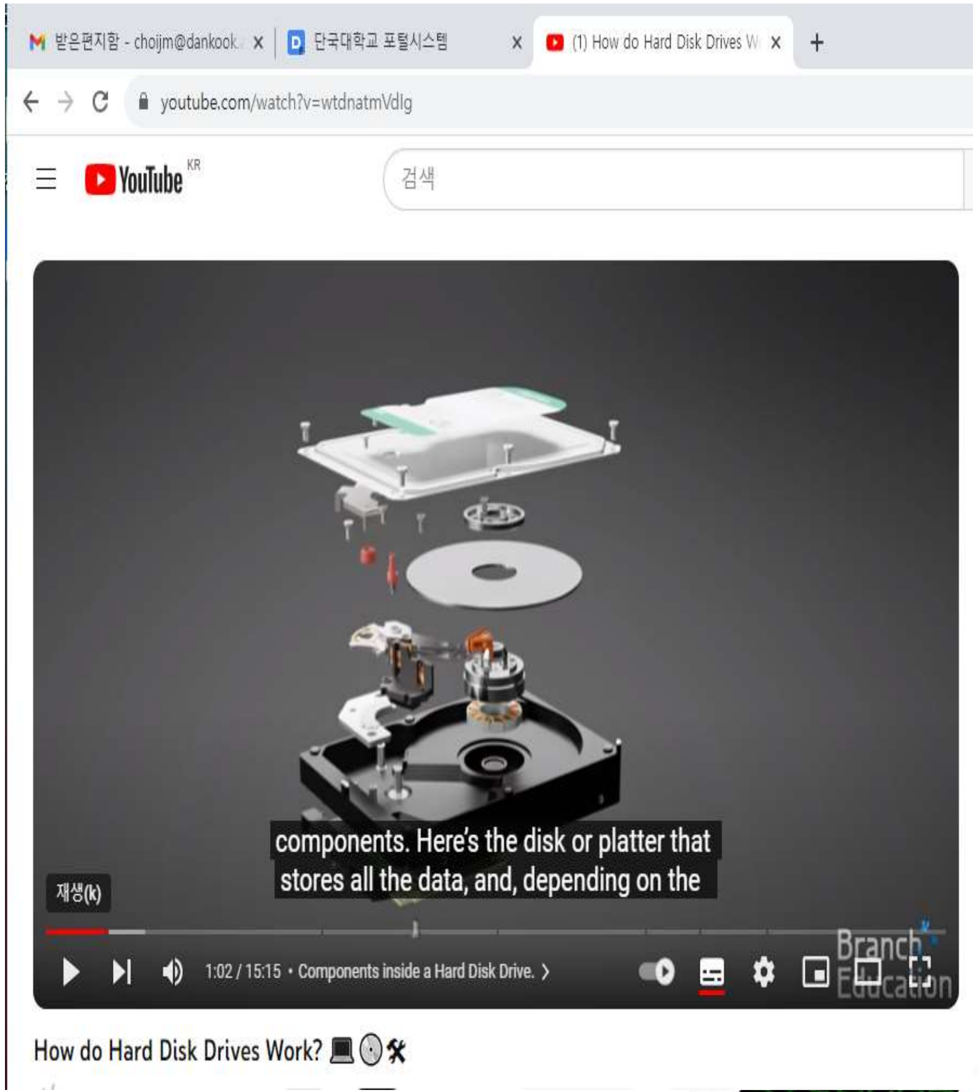
- Disk access
  - ✓ Disk behaviors (from youtube)



- Previous Lecture Information

시스템프로그래밍 (System Programming)

- 강의 자료 (Lecture Notes)
  - Lecture Note 0: *Course overview*
  - Lecture Note 1: *What is System Programming?*
  - Lecture Note 2: *Programming Environment*
  - Lecture Note 3: *File Programming*
  - Lecture Note 4: *Process Structure*
  - Lecture Note 5: *Process Programming*
  - Lecture Note 6: *IA Assembly Programming*
  - Lecture Note 7: *IA History and Features*
  - Lecture Note 8: *Optimization and Monitoring*
  - Lecture Note 9: *Assembler*
  - Lecture Note 10: *Linker, Dubbger and Tools*
- 강의 교재
  - Textbook1: *Computer Systems: A Programmer's Perspective (3rd Edition)* by R. Bryant and D. O'Hallaron
    - *Lecture site* of the "Computer Systems: A Programmer's Perspective"
    - *Chapter 1* of the "Computer Systems: A Programmer's Perspective (2nd edition)"
  - Textbook2: *The Linux Programming Interface* by M. Kerrisk
- 강의 관련 자료
  - *Advanced Programming in the UNIX Environments by R. Stevens. Addison Wesley*
  - *리눅스 커널 내부구조 by 백승재, 최종무, 아티오*
  - *Linux System Programming: Talking Directly to the Kernel and C Library by R. Love. O'Reilly*
  - *유닉스/리눅스 프로그래밍 필수 유틸리티 by 백창우, 한빛미디어*
  - *Intel 64 and IA-32 Architecture Software Developer's Manual*
  - *ARM System-on-Chip Architecture (2nd Edition)* by S. Furber
  - The UNIX time-sharing system: *UNIX paper*
  - *RAG 기반 운영체제 교육*
  - *Dr Lee's Kaggle demo*
  - *How do Hard Disk Drives Work?*
  - *Inside of a Hard Disk*
  - *Concept of Pipeline*
  - *Memory Address*
  - *GNU GCC*
  - *GNU Assembler*



받은편지함 - choijm@dankook
단국대학교 포털시스템
(1) How do Hard Disk Drives W
youtube.com/watch?v=wtdnatmVdlg
YouTube KR
검색
재생(k)
1:02 / 15:15 • Components inside a Hard Disk Drive.
Branch Education

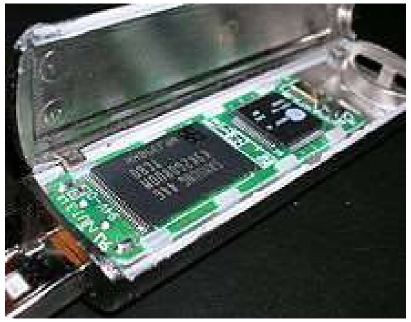
components. Here's the disk or platter that stores all the data, and, depending on the

How do Hard Disk Drives Work?

Disk structure (4/4, Optional)

- Disk vs. Flash memory

vs



- ✓ No mechanical part (fast, lightweight)
- ✓ Overwrite limitation (erase before write)
- ✓ Read/Write vs. Erase granularity
- ✓ Endurance, Disturbance, Retention error
- ✓ Types: SLC, MLC, TLC, QLC, …

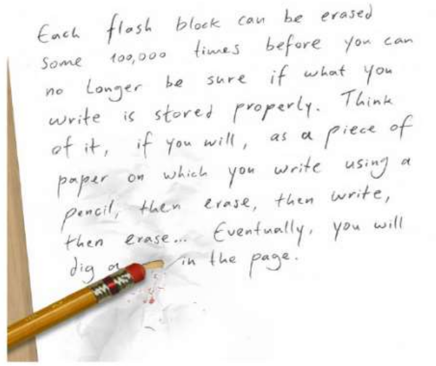
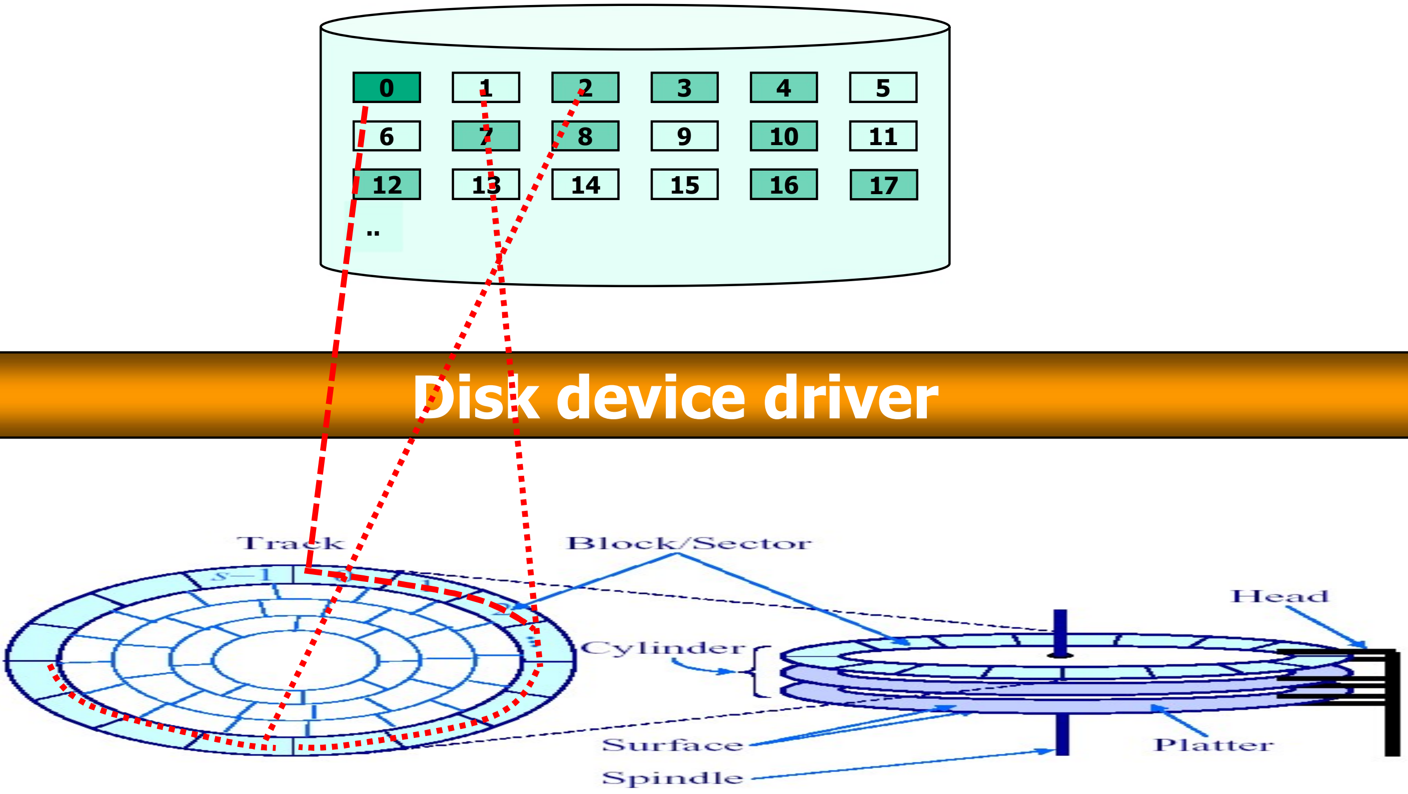

Each flash block can be erased some 100,000 times before you can no longer be sure if what you write is stored properly. Think of it, if you will, as a piece of paper on which you write using a pencil, then erase, then write, then erase... Eventually, you will dig a [illegible] in the page.

Figure 1: Flash, like paper, can only be erased so many times before it gets used up.

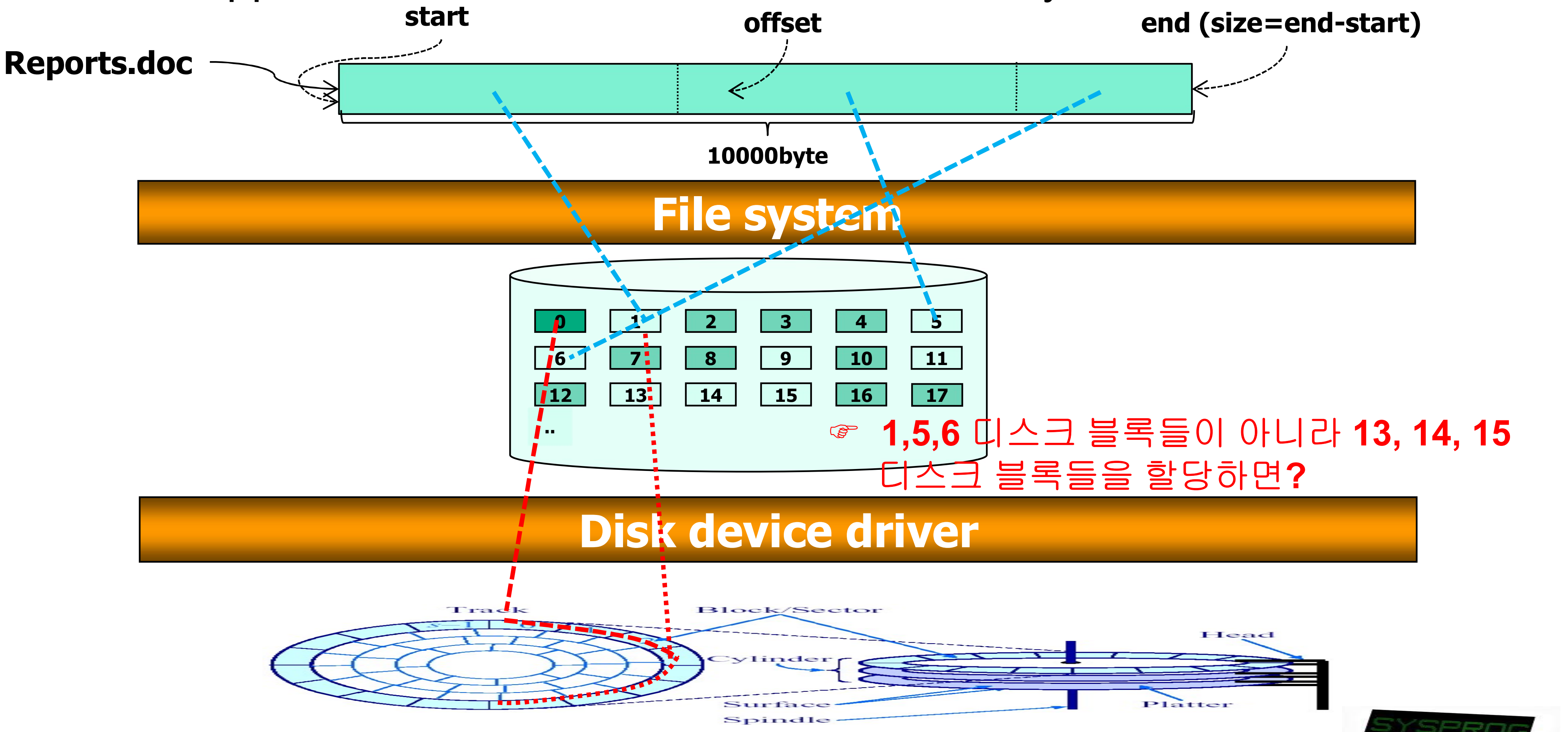
System programs for Disk (1/7)

- Disk device driver
  - ✓ Abstract disk as a logical disk (a collection of disk blocks)
    - The size of a disk block is the same as that of page frame (4 or 8KB)
  - ✓ Disk command handling (ATA command: type, start, size, device, …)
  - ✓ Disk initialization, scheduling, error handling, …



System programs for Disk (2/7)

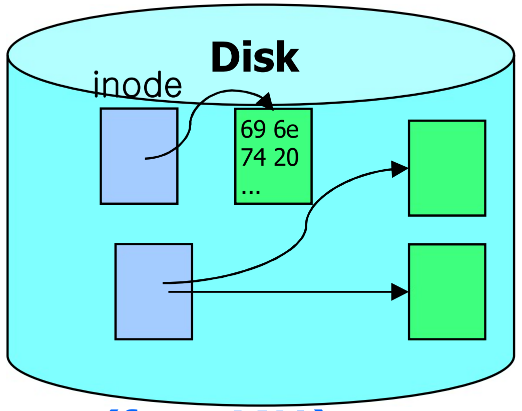
- File system
  - ✓ Support file abstraction: stream of bytes
  - ✓ Associate a file with disk blocks (inode, FAT)
  - ✓ Support file attribute/access control, directory, …



☞ 1,5,6 디스크 블록들이 아니라 13, 14, 15 디스크 블록들을 할당하면?

System programs for Disk (3/7)

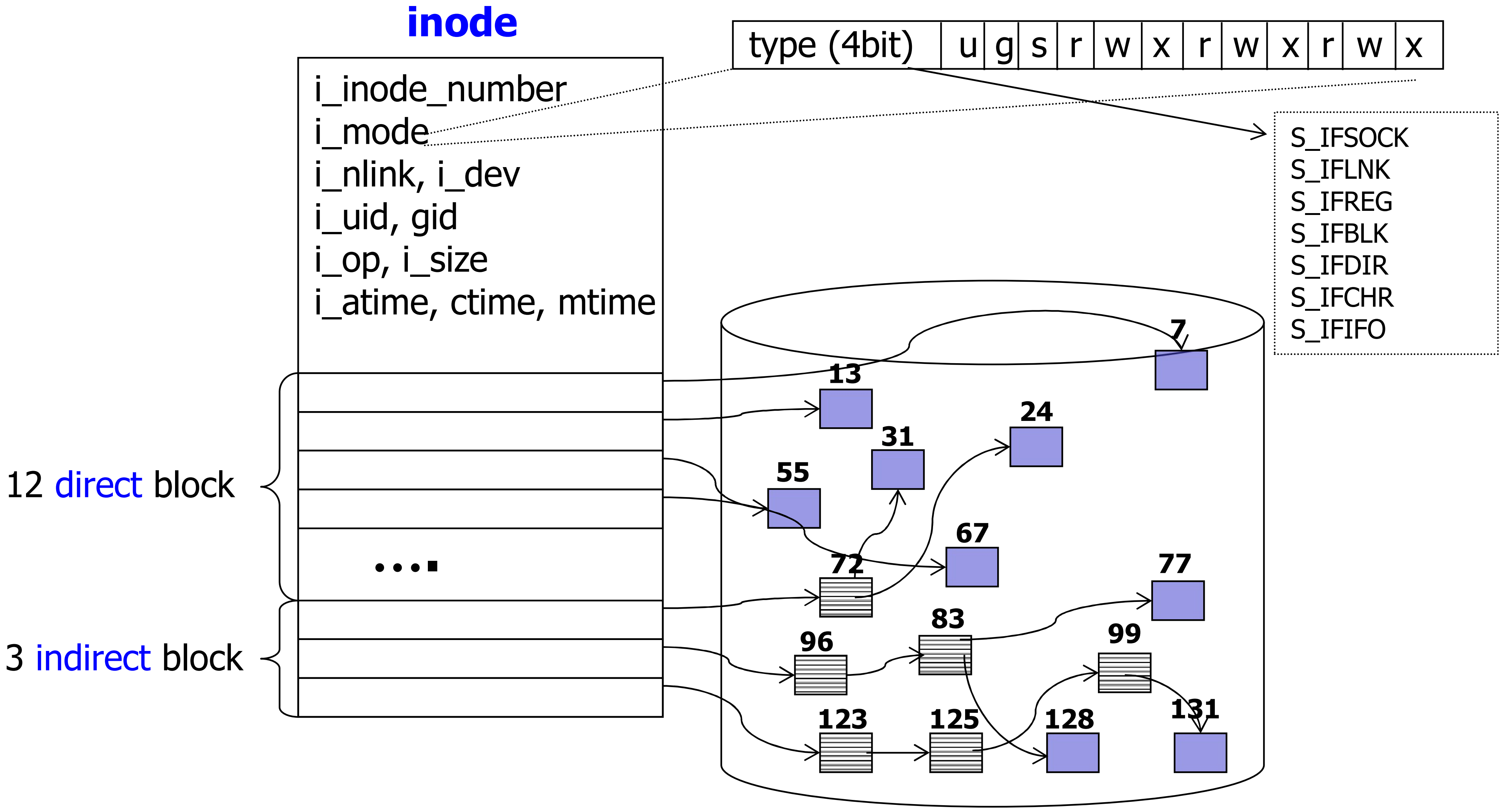
- File system
  - ✓ inode concept
    - An object for managing a file in a file system (metadata)
    - Used by various file systems such as UFS, FFS, Ext2/3/4, LFS, …
    - Maintain information for a file (e.g. “ls –l”)
      - file size
      - locations of disk blocks for a file
      - file owner, access permission
      - time information
      - file type: regular, directory, device, pipe, socket, …
    - Stored in disk
    - Constructed when a file is created
    - Accessed when it is opended



(from LN1)

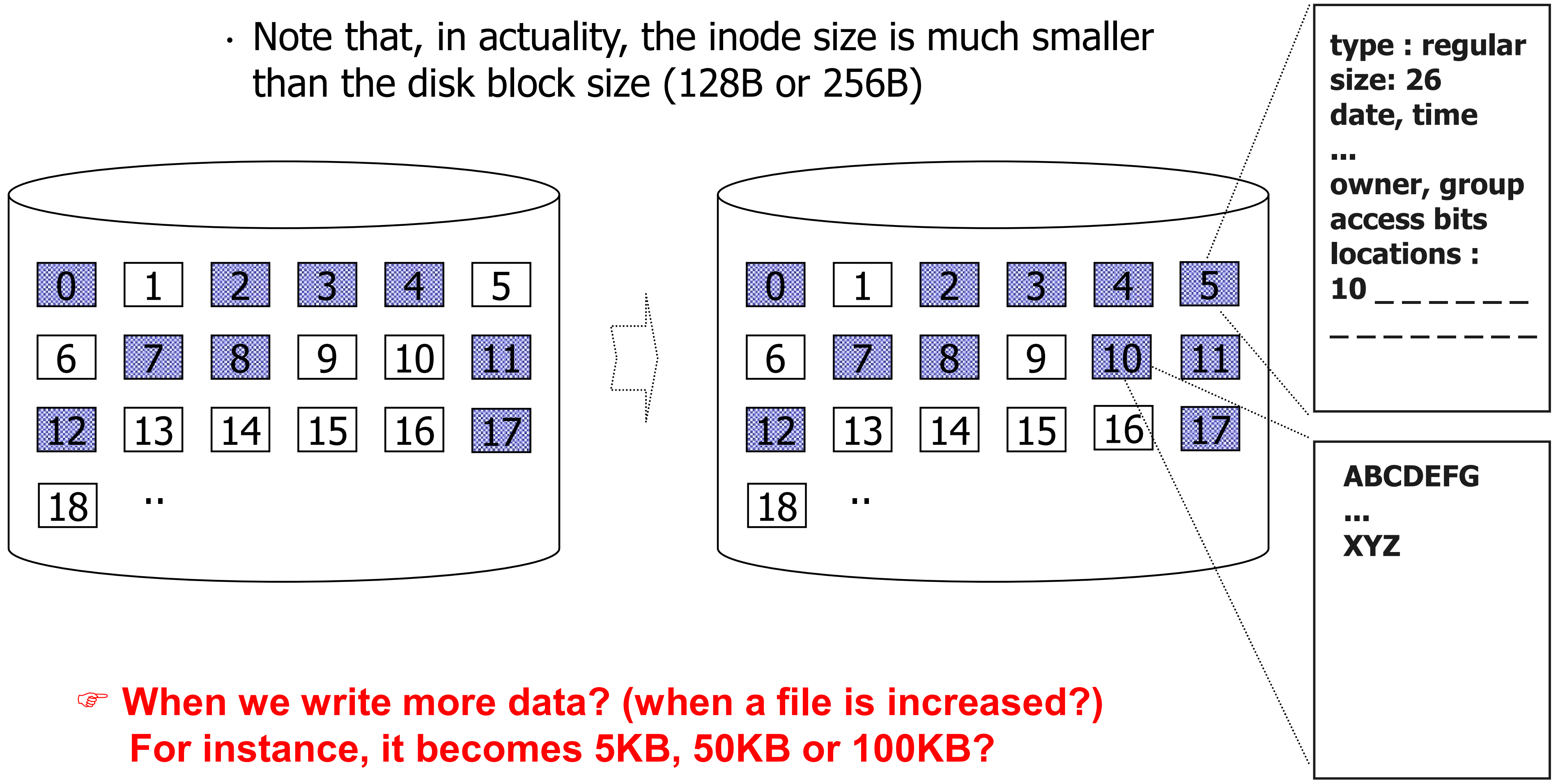
System programs for Disk (4/7)

- File system
  - ✓ inode structure



System programs for Disk (5/7)

- File system
  - ✓ inode example
    - When we create a new file, named “alphabet.txt”, whose contents include “AB…Z”.
      - Note that, in actuality, the inode size is much smaller than the disk block size (128B or 256B)

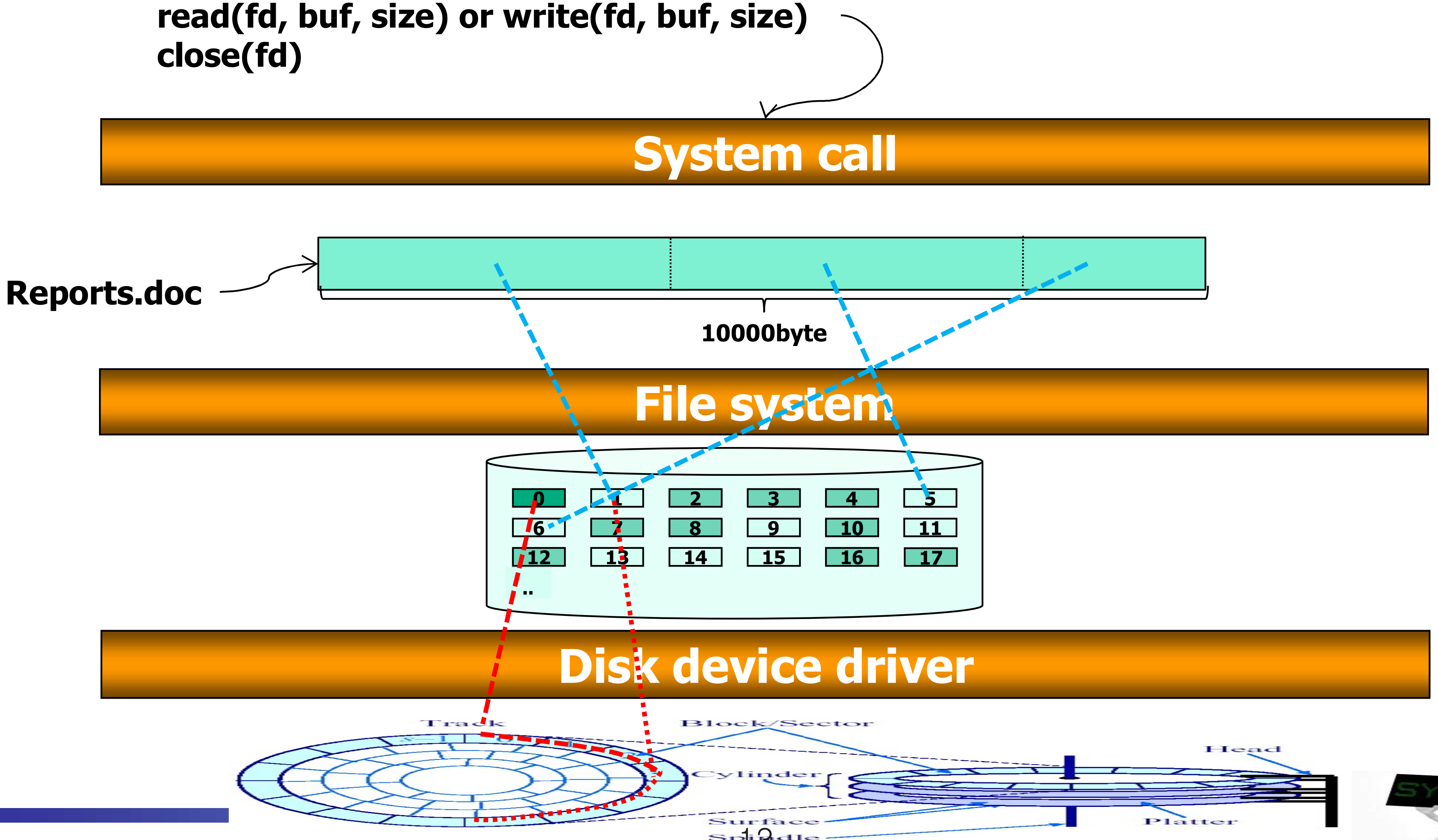


☞ **When we write more data? (when a file is increased?) For instance, it becomes 5KB, 50KB or 100KB?**

System programs for Disk (6/7)

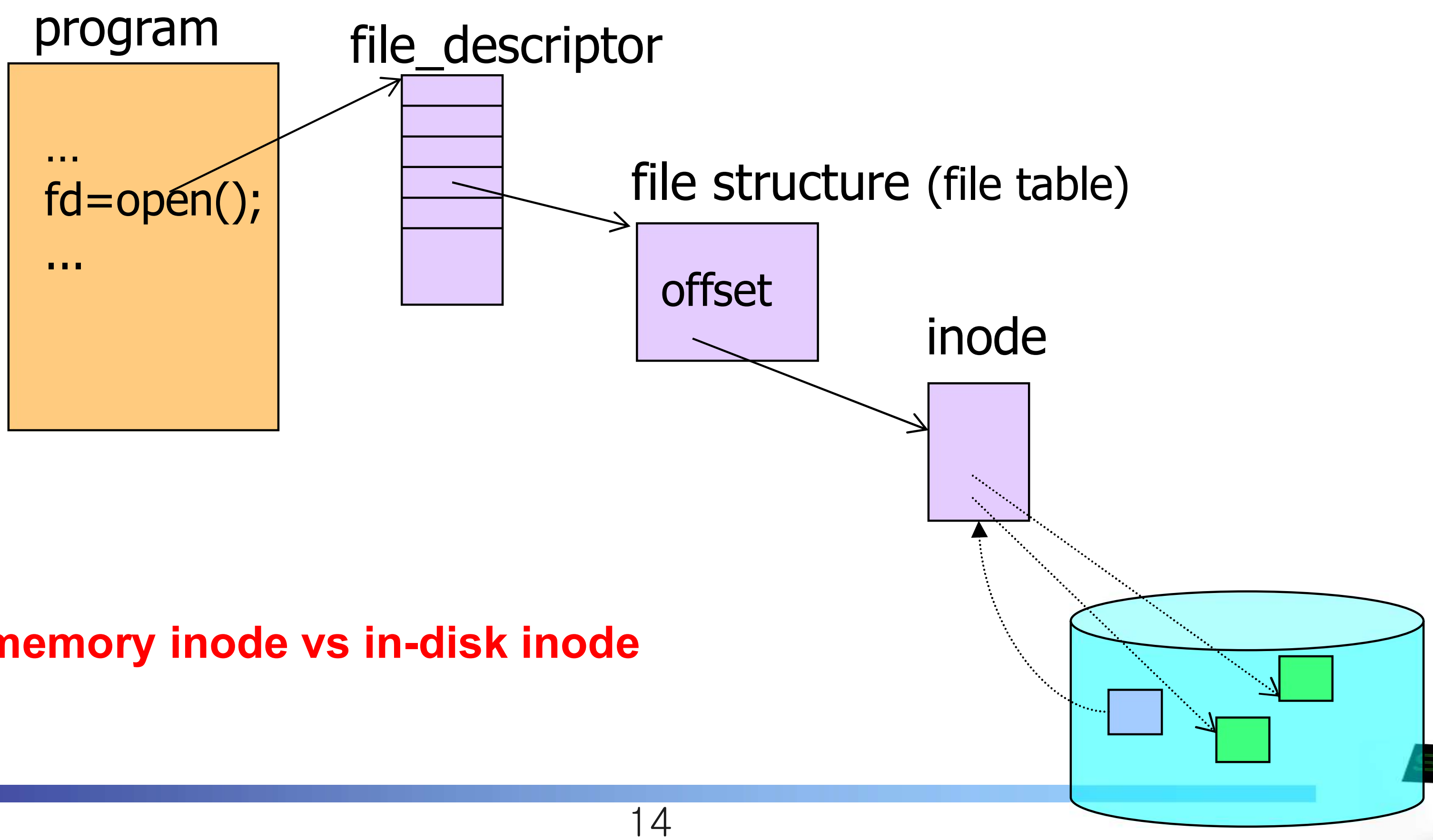
- System call
  - ✓ Support interfaces such as open(), read(), write(), close(), …

fd=open("Reports.doc", …)
read(fd, buf, size) or write(fd, buf, size)
close(fd)



System programs for Disk (7/7)

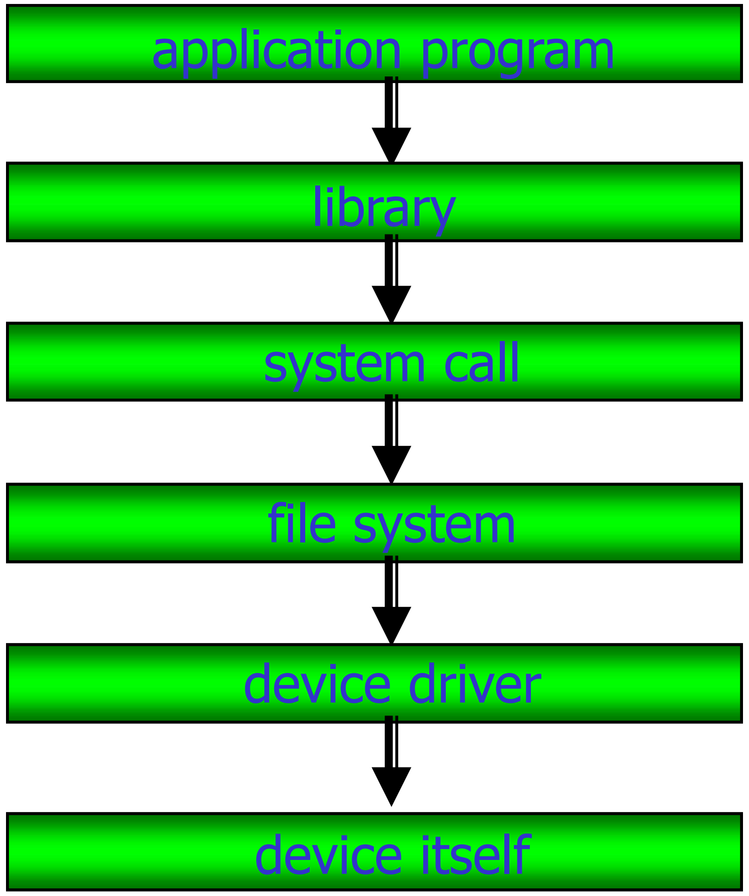
- System call
  - ✓ Use fd (file descriptor) instead of file name (for efficiency)
    - fd: object to point out a file in kernel
    - return value of the open() system call
    - used by the following read(), write(), …, close() system calls
    - fd is connected into inode through various kernel objects (file table)



☞ **in-memory inode vs in-disk inode**

Layered Architecture for Abstraction

- Revisit LN1



File Programming: Basic (1/11)

- Practice 1: read data from an existing file

```
/* file_test1.c: read data from a file, by choijm. choijm@dankook.ac.kr*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#define MAX_BUF    16
char fname[] = "alphabet.txt";

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

    fd = open(fname, O_RDONLY);
    if (fd < 0) {
            printf("Can't open %s file with errno %d₩n", fname, errno);
            exit(-1);
    }
    size = read(fd, buf, MAX_BUF);
    if (size < 0) {
            printf("Can't read from file %s, size = %d₩n", fname, size);
            exit(-1);
    }
    else
            printf("size of read data is %d₩n", size);
    close(fd);
}
```

Refer to next slide (Syntax)

Inform the cause when an error occurs
cf) Error handling is quite important!!

File Programming: Basic (2/11)

- Syntax of the open() and read() system call

int open(const char *pathname, int flags, [mode_t mode])

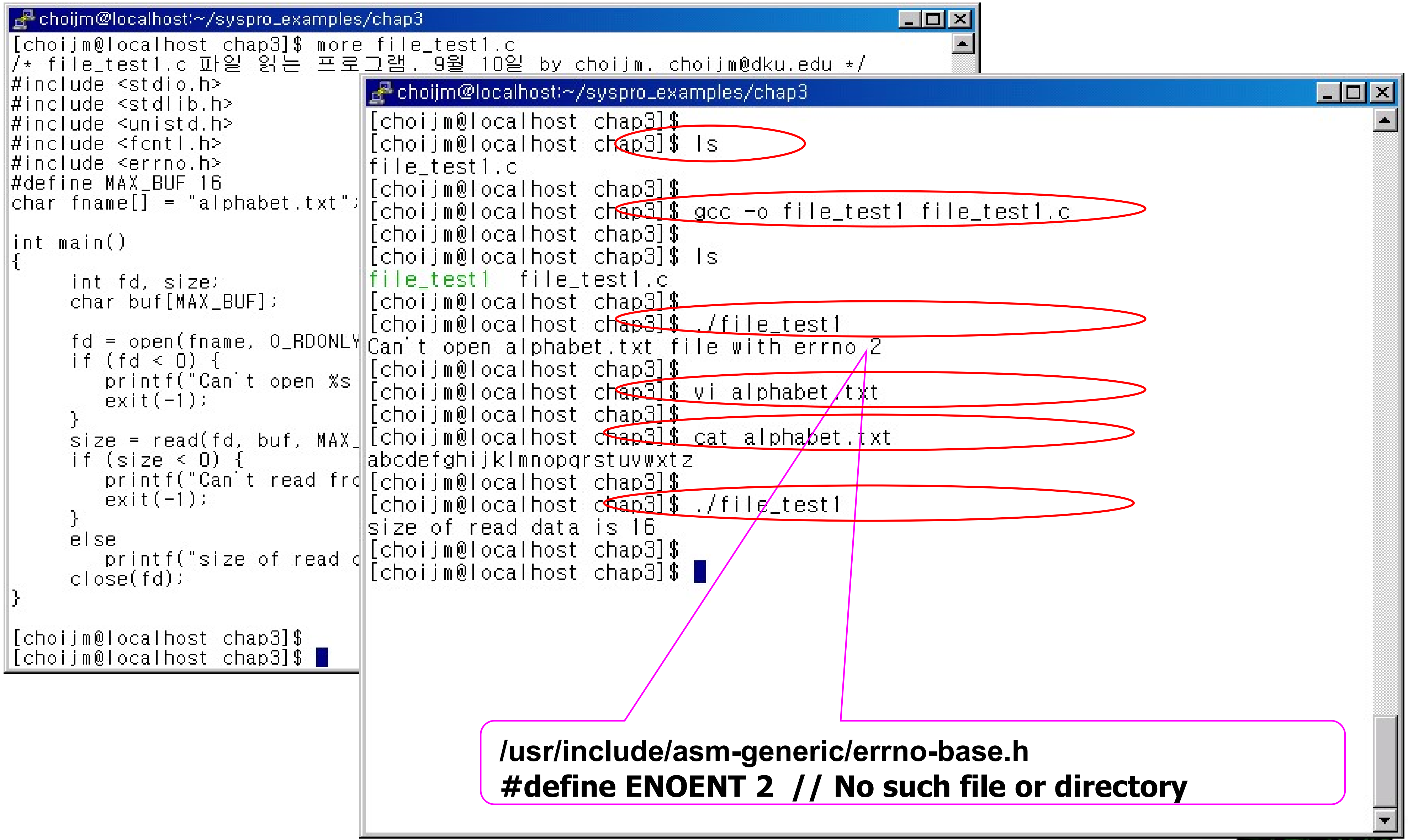
- ✓ pathname : absolute path or relative path
- ✓ flags (see: /usr/include/asm/fcntl.h or Chapter 4.3 in the LPI)
  - O_RDONLY, O_WRONLY, O_RDWR
  - O_CREAT, O_EXCL
  - O_TRUNC, O_APPEND
  - O_NONBLOCK, O_SYNC
  - …
- ✓ mode
  - meaningful with the O_CREAT flag
  - file access mode (S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, …, S_IROTH, …)
- ✓ return value
  - file descriptor if success
  - -1 if fail

int read(int fd, char *buf, int size) // same as the write(fd, buf, size)

- ✓ fd: file descriptor (return value of open())
- ✓ buf: memory space for keeping data
- ✓ size: request size
- ✓ return value
  - read size
  - -1 if fail

File Programming: Basic (3/11)

- Practice 1: execution results



choijm@localhost:~/syspro_examples/chap3

```
[choijm@localhost chap3]$ more file_test1.c
/* file_test1.c 파일 읽는 프로그램. 9월 10일 by choijm. choijm@dku.edu */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#define MAX_BUF 16
char fname[] = "alphabet.txt";

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

     fd = open(fname, O_RDONLY
     if (fd < 0) {
        printf("Can't open %s
        exit(-1);
     }
     size = read(fd, buf, MAX_
     if (size < 0) {
        printf("Can't read fro
        exit(-1);
     }
     else
        printf("size of read c
     close(fd);
}

[choijm@localhost chap3]$
[choijm@localhost chap3]$
```

choijm@localhost:~/syspro_examples/chap3

```
[choijm@localhost chap3]$
[choijm@localhost chap3]$ ls
file_test1.c
[choijm@localhost chap3]$
[choijm@localhost chap3]$ gcc -o file_test1 file_test1.c
[choijm@localhost chap3]$
[choijm@localhost chap3]$ ls
file_test1  file_test1.c
[choijm@localhost chap3]$
[choijm@localhost chap3]$ ./file_test1
Can't open alphabet.txt file with errno 2
[choijm@localhost chap3]$
[choijm@localhost chap3]$ vi alphabet.txt
[choijm@localhost chap3]$
[choijm@localhost chap3]$ cat alphabet.txt
abcdefghijklmnopqrstuvwxtz
[choijm@localhost chap3]$
[choijm@localhost chap3]$ ./file_test1
size of read data is 16
[choijm@localhost chap3]$
[choijm@localhost chap3]$
```

/usr/include/asm-generic/errno-base.h
#define ENOENT 2 // No such file or directory

File Programming: Basic (4/11)

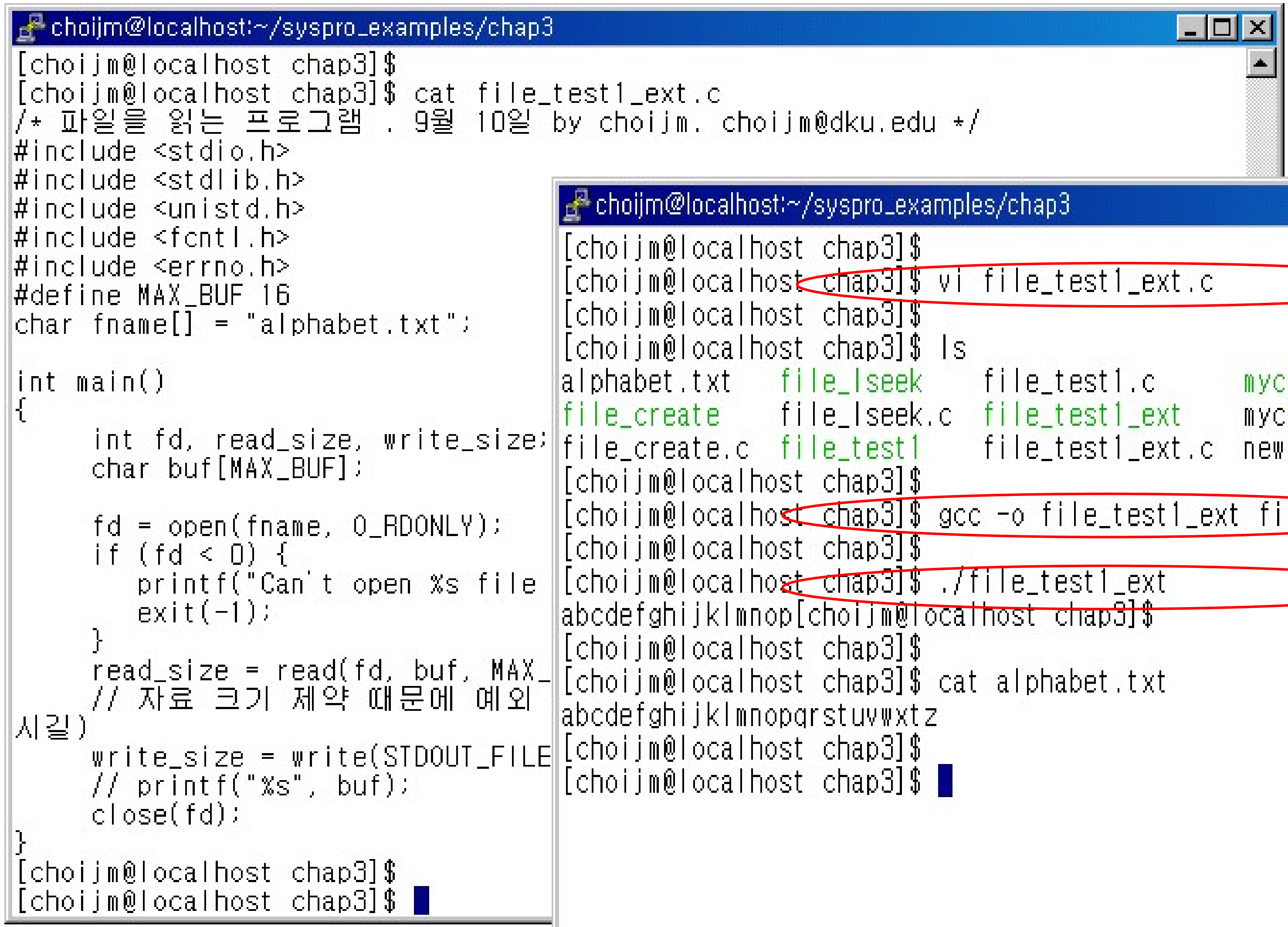
- Practice 2: extend the practice 1 so that it displays the read data on terminal

```
/* file_test1_ext.c: read data from a file and display them, by choijm. choijm@dku.edu*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#define MAX_BUF    16
char fname[] = "alphabet.txt";

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

    fd = open(fname, O_RDONLY);
    if (fd < 0) {
            printf("Can't open %s file with errno %d₩n", fname, errno);
            exit(-1);
    }
    read_size = read(fd, buf, MAX_BUF);
    // Due to the slide limit, I omit the error handling code (But, students must implement it)
    write_size = write(STDOUT_FILENO, buf, read_size);
    close(fd);
}
```

```
/usr/include/unistd.h 참조
#define STDIN_FILENO     0   // Standard input
#define STDOUT_FILENO    1   // Standard output
#define STDERR_FILENO    2   // Standard error
```

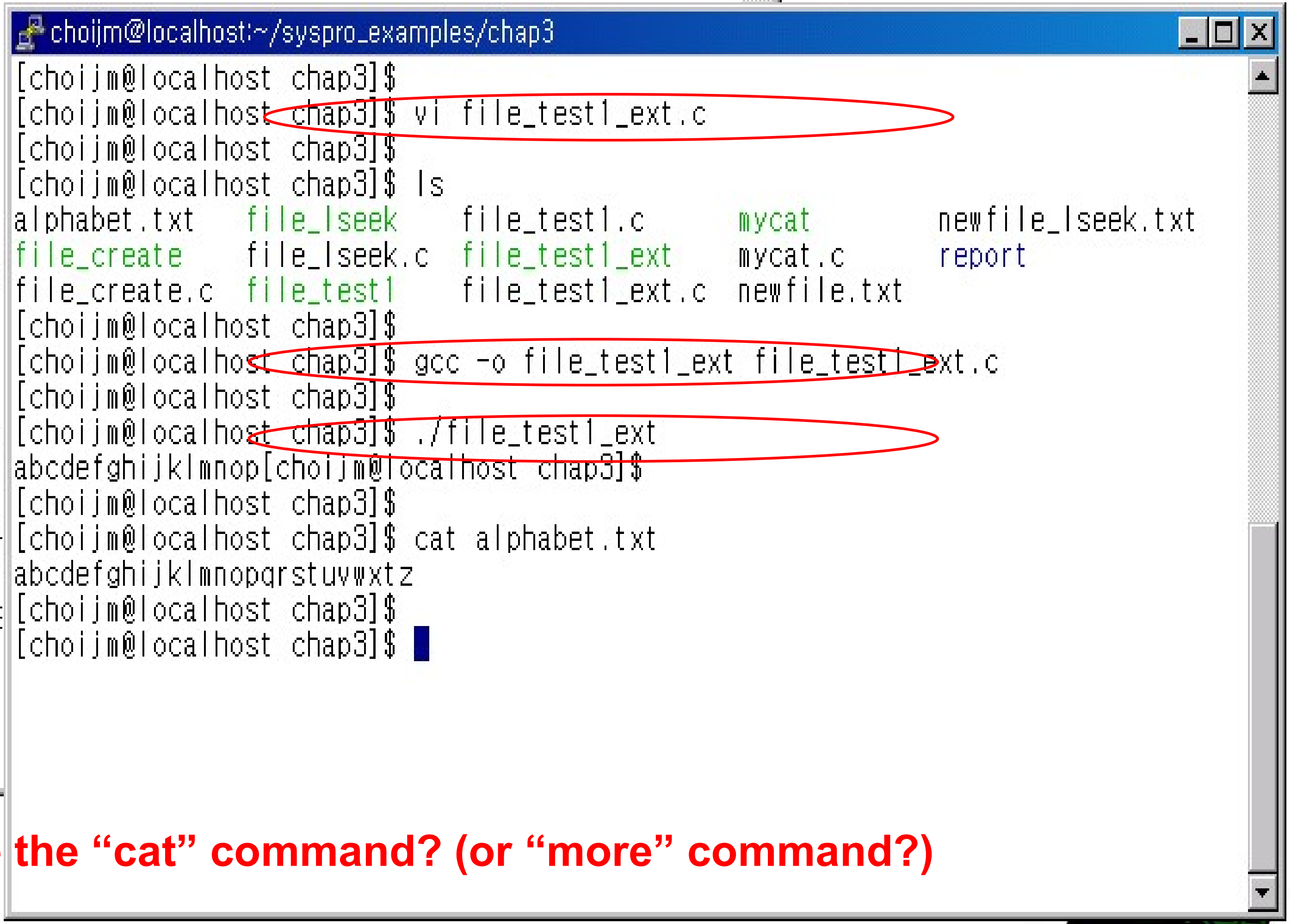
File Programming: Basic (5/11)

- Practice 2: execution results



choijm@localhost:~/syspro_examples/chap3

```
[choijm@localhost chap3]$
[choijm@localhost chap3]$ cat file_test1_ext.c
/* 파일을 읽는 프로그램 . 9월 10일 by choijm. choijm@dku.edu */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#define MAX_BUF 16
char fname[] = "alphabet.txt";

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

     fd = open(fname, O_RDONLY);
     if (fd < 0) {
        printf("Can't open %s file
        exit(-1);
     }
     read_size = read(fd, buf, MAX_
     // 자료 크기 제약 때문에 예외
시킴)
     write_size = write(STDOUT_FILE
     // printf("%s", buf);
     close(fd);
}
[choijm@localhost chap3]$
[choijm@localhost chap3]$
```



choijm@localhost:~/syspro_examples/chap3

```
[choijm@localhost chap3]$
[choijm@localhost chap3]$ vi file_test1_ext.c
[choijm@localhost chap3]$
[choijm@localhost chap3]$ ls
alphabet.txt   file_lseek    file_test1.c      mycat        newfile_lseek.txt
file_create    file_lseek.c  file_test1_ext    mycat.c      report
file_create.c  file_test1    file_test1_ext.c  newfile.txt
[choijm@localhost chap3]$
[choijm@localhost chap3]$ gcc -o file_test1_ext file_test1_ext.c
[choijm@localhost chap3]$
[choijm@localhost chap3]$ ./file_test1_ext
abcdefghijklmnop[choijm@localhost chap3]$
[choijm@localhost chap3]$
[choijm@localhost chap3]$ cat alphabet.txt
abcdefghijklmnopqrstuvwxtz
[choijm@localhost chap3]$
[choijm@localhost chap3]$
```

☞ **Can we make the “cat” command? (or “more” command?)**

File Programming: Basic (6/11)

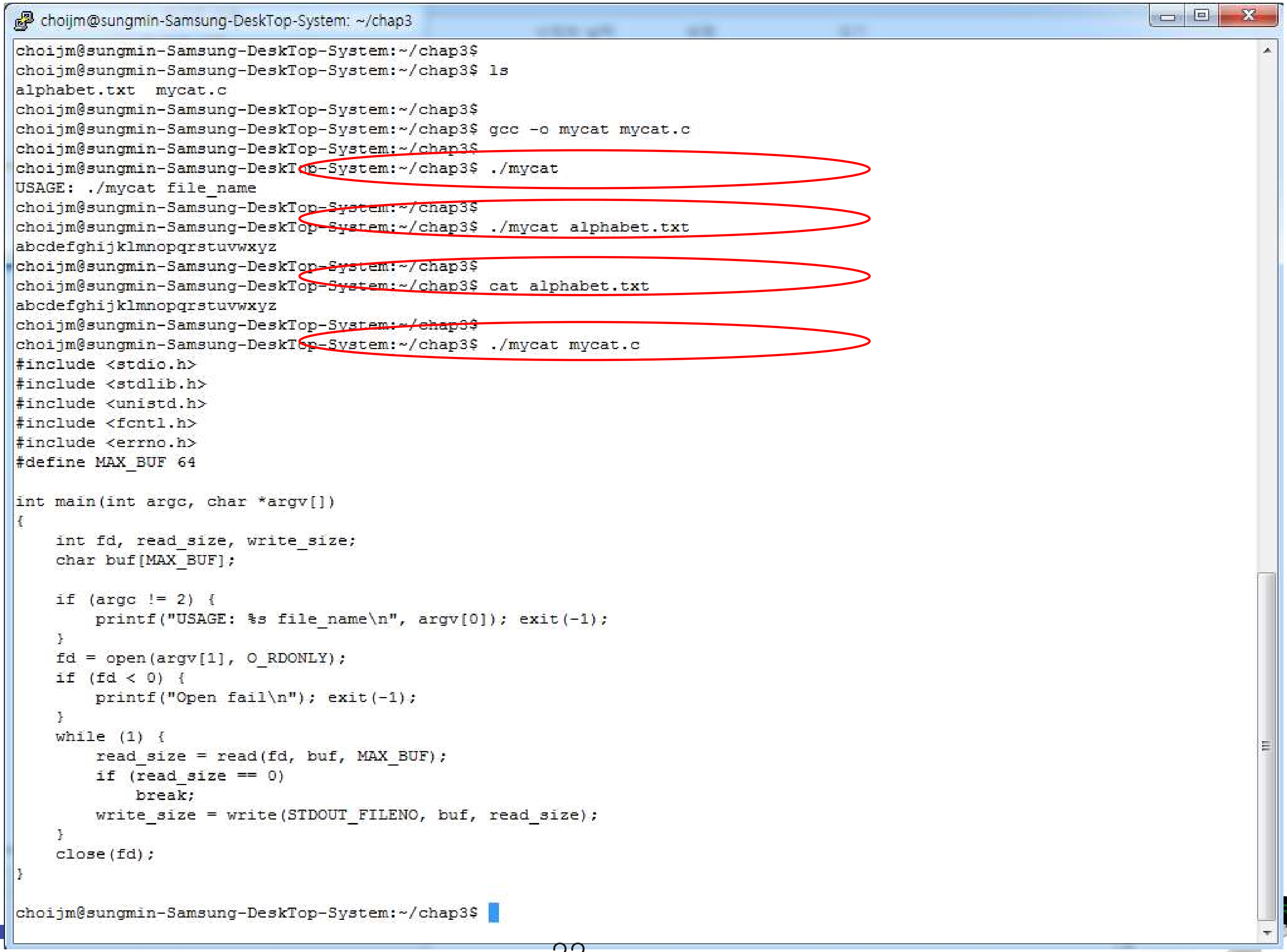
- Practice 3: make a “mycat” command (with argc, argv)

```
/* mycat program, by choijm. choijm@dku.edu */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#define MAX_BUF 64

int main(int argc, char *argv[])
{
    int fd, read_size, write_size;
    char buf[MAX_BUF];

    if (argc != 2) {
        printf("USAGE: %s file_name₩n", argv[0]);  exit(-1);
    }
    fd = open(argv[1], O_RDONLY);
    if (fd < 0) {
        // open error handling
    }
    while (1) {
        read_size = read(fd, buf, MAX_BUF);
         if (read_size == 0)
                break;
        write_size = write(STDOUT_FILENO, buf, read_size);
    }
    close(fd);
}
```

Command Convention

File Programming: Basic (7/11)

- Practice 3: execution results



```
choijm@sungmin-Samsung-DeskTop-System:~/chap3$
choijm@sungmin-Samsung-DeskTop-System:~/chap3$ ls
alphabet.txt  mycat.c
choijm@sungmin-Samsung-DeskTop-System:~/chap3$
choijm@sungmin-Samsung-DeskTop-System:~/chap3$ gcc -o mycat mycat.c
choijm@sungmin-Samsung-DeskTop-System:~/chap3$
choijm@sungmin-Samsung-DeskTop-System:~/chap3$ ./mycat
USAGE: ./mycat file_name
choijm@sungmin-Samsung-DeskTop-System:~/chap3$
choijm@sungmin-Samsung-DeskTop-System:~/chap3$ ./mycat alphabet.txt
abcdefghijklmnopqrstuvwxyz
choijm@sungmin-Samsung-DeskTop-System:~/chap3$
choijm@sungmin-Samsung-DeskTop-System:~/chap3$ cat alphabet.txt
abcdefghijklmnopqrstuvwxyz
choijm@sungmin-Samsung-DeskTop-System:~/chap3$
choijm@sungmin-Samsung-DeskTop-System:~/chap3$ ./mycat mycat.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#define MAX_BUF 64

int main(int argc, char *argv[])
{
    int fd, read_size, write_size;
    char buf[MAX_BUF];

    if (argc != 2) {
        printf("USAGE: %s file_name\n", argv[0]); exit(-1);
    }
    fd = open(argv[1], O_RDONLY);
    if (fd < 0) {
        printf("Open fail\n"); exit(-1);
    }
    while (1) {
        read_size = read(fd, buf, MAX_BUF);
        if (read_size == 0)
            break;
        write_size = write(STDOUT_FILENO, buf, read_size);
    }
    close(fd);
}

choijm@sungmin-Samsung-DeskTop-System:~/chap3$
```

File Programming: Basic (8/11)

- Practice 4: create a new file

```
/* file_create.c: create a new file, by choijm. choijm@dku.edu */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#define MAX_BUF 64
char fname[] = "newfile.txt";
char dummy_data[]="abcdefg₩n";

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

    fd = open(fname, O_RDWR | O_CREAT | O_EXCL, 0664);
    if (fd < 0) {
        printf("Can't create %s file with errno %d₩n", fname, errno); exit(1);
    }
    write_size = write(fd, dummy_data, sizeof(dummy_data));
    printf("write_size = %d₩n", write_size);
    close(fd);

    fd = open(fname, O_RDONLY);
    read_size = read(fd, buf, MAX_BUF);
    printf("read_size = %d₩n", read_size);
    write_size = write(STDOUT_FILENO, buf, read_size);
    close(fd);
}
```

If we rerun this program?

O_CREAT or creat()

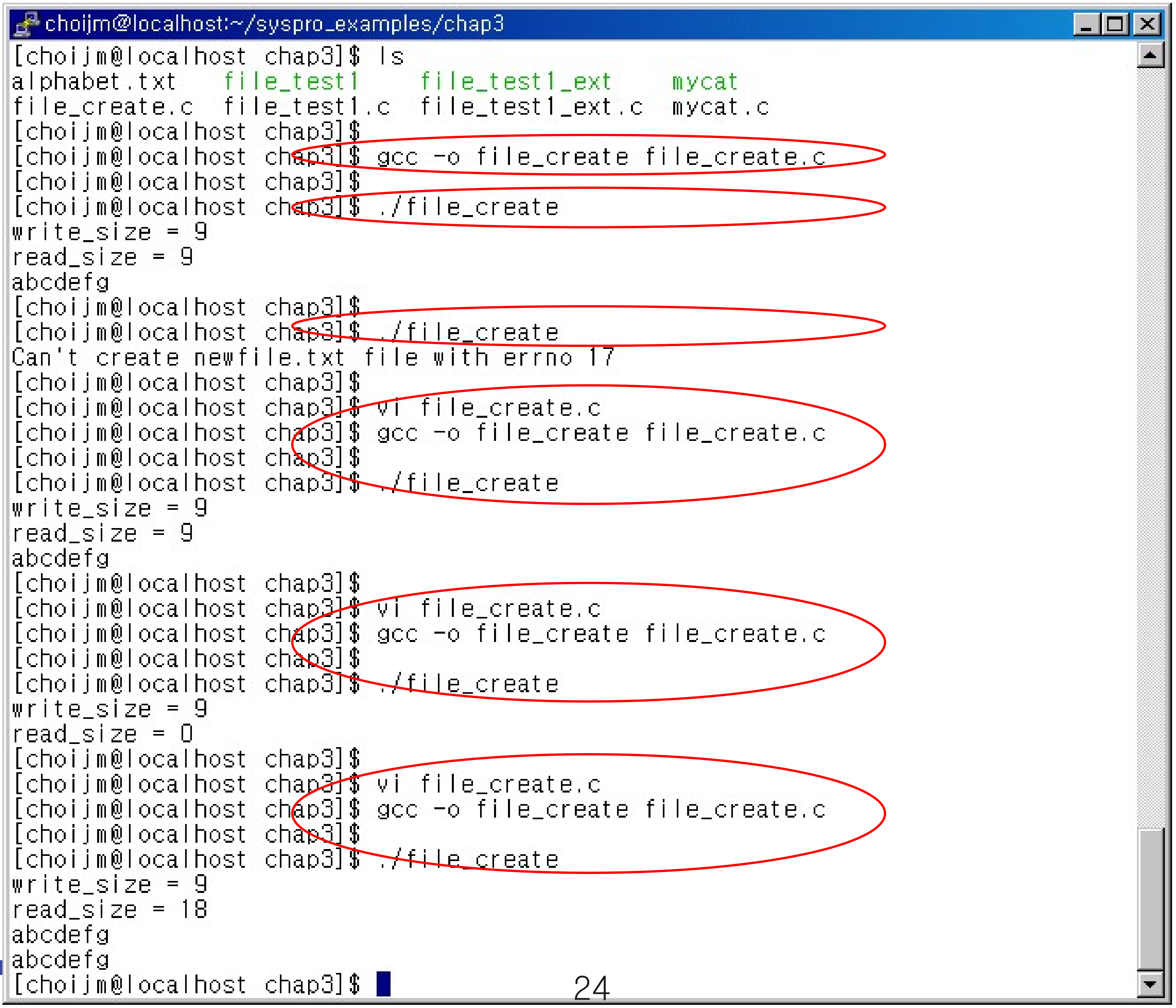
If we rerun without the O_EXCL flag?

If we want to write data at the end of this file?

If we comment out these close() and open() statements?

File Programming: Basic (9/11)

- Practice 4: execution results



choijm@localhost:~/syspro_examples/chap3

```
[choijm@localhost chap3]$ ls
alphabet.txt   file_test1     file_test1_ext     mycat
file_create.c  file_test1.c   file_test1_ext.c   mycat.c
[choijm@localhost chap3]$
[choijm@localhost chap3]$ gcc -o file_create file_create.c
[choijm@localhost chap3]$
[choijm@localhost chap3]$ ./file_create
write_size = 9
read_size = 9
abcdefg
[choijm@localhost chap3]$
[choijm@localhost chap3]$ ./file_create
Can't create newfile.txt file with errno 17
[choijm@localhost chap3]$
[choijm@localhost chap3]$ vi file_create.c
[choijm@localhost chap3]$ gcc -o file_create file_create.c
[choijm@localhost chap3]$
[choijm@localhost chap3]$ ./file_create
write_size = 9
read_size = 9
abcdefg
[choijm@localhost chap3]$
[choijm@localhost chap3]$ vi file_create.c
[choijm@localhost chap3]$ gcc -o file_create file_create.c
[choijm@localhost chap3]$
[choijm@localhost chap3]$ ./file_create
write_size = 9
read_size = 0
[choijm@localhost chap3]$
[choijm@localhost chap3]$ vi file_create.c
[choijm@localhost chap3]$ gcc -o file_create file_create.c
[choijm@localhost chap3]$
[choijm@localhost chap3]$ ./file_create
write_size = 9
read_size = 18
abcdefg
abcdefg
[choijm@localhost chap3]$
```

File Programming: Basic (10/11)

- Practice 5: want to read "d" from a file whose contents are "abcdefg"
  - ✓ Using lseek()

off_t lseek(int fd, off_t offset, int whence)

- ✓ fd : file descriptor
- ✓ offset : offset position
- ✓ whence (/usr/include/unistd.h)
  - SEEK_SET : New offset is set to offset bytes.
  - SEEK_CUR: New offset is set to its current location plus offset bytes.
  - SEEK_END: New offset is set to the size of the file plus offset bytes
- ✓ return value
  - new offset if success
  - -1 if fail

Negative value is allowed

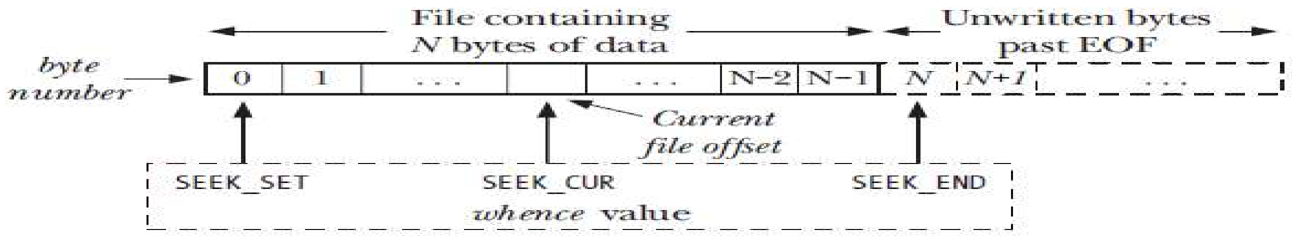


Figure 4-1: Interpreting the *whence* argument of *lseek()*

☞ **sequential access vs. random access**

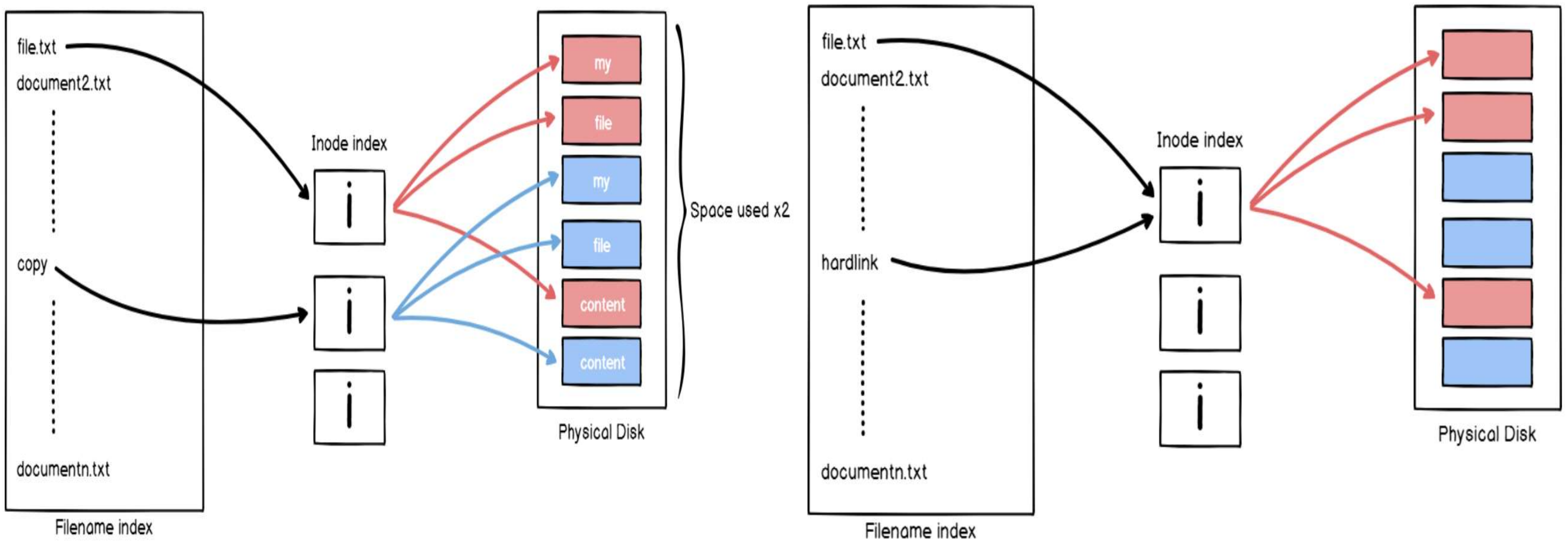
File Programming: Basic (11/11)

- Practice 5: want to read “d” from a file whose contents are “abcdefg”

```
/* file_lseek.c: lseek example, by choijm. choijm@dku.edu */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#define MAX_BUF 64
char fname[] = "newfile_lseek.txt";
char dummy_data[]="abcdefg₩n";

int main()
{
   int fd, write_size, read_size, new_offset;
   char buf[MAX_BUF];

   fd = open(fname, O_RDWR | O_CREAT | O_EXCL, 0664);
   write_size = write(fd, dummy_data, sizeof(dummy_data)); printf("write_size = %d₩n", write_size);
   close(fd);

   fd = open(fname, O_RDONLY);
   new_offset = lseek(fd, 3, SEEK_SET);
   read_size = read(fd, buf, MAX_BUF); printf("read_size = %d₩n", read_size);
   write_size = write(STDOUT_FILENO, buf, read_size);
   close(fd);
}
```

File Programming: Advanced (1/6)

- Other system calls related to file
  - ✓ creat() // same as open() with flag **O_WRONLY | O_CREAT | O_TRUNC**
  - ✓ mkdir(), readdir(), rmdir()
  - ✓ pipe()
  - ✓ mknod()
  - ✓ link(), unlink()



(Source: https://devconnected.com/understanding-hard-and-soft-links-on-linux/)

File Programming: Advanced (2/6)

- Other system calls related to file
  - ✓ dup(), dup2()
  - ✓ stat(), fstat()
  - ✓ chmod(), fchmod()
  - ✓ ioctl(), fcntl()
  - ✓ sync(), fsync()

Figure 10.11
Typical kernel data structures for open files. In this example, two descriptors reference distinct files. There is no sharing.

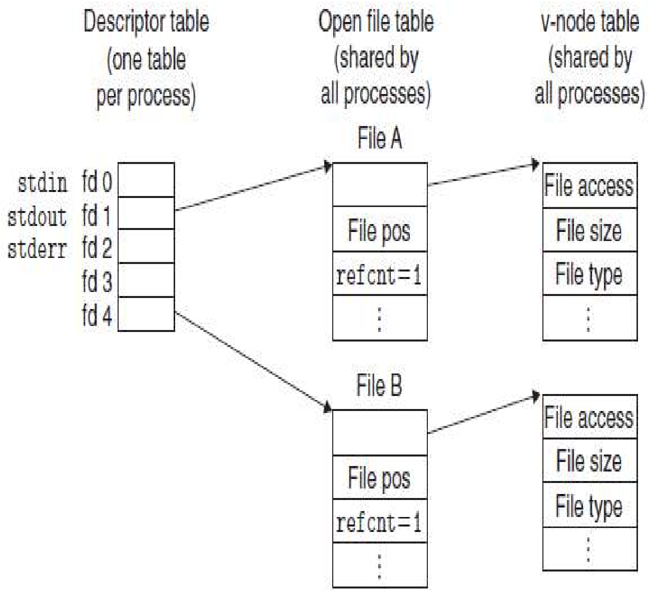
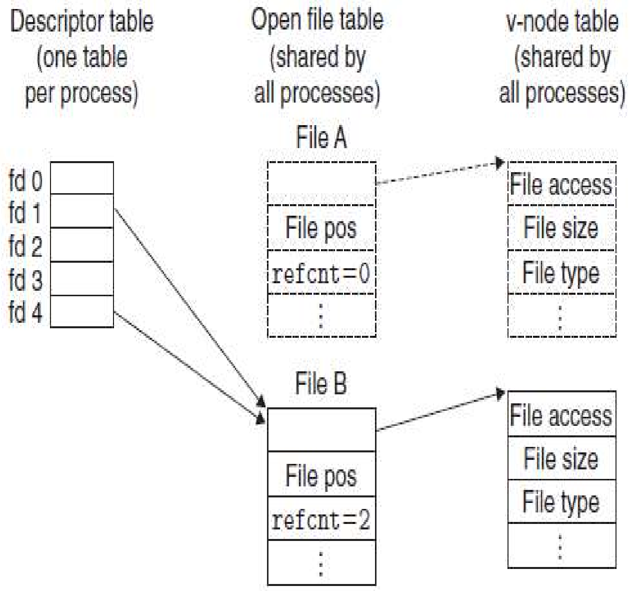


Figure 10.14
Kernel data structures after redirecting standard output by calling dup2(4,1). The initial situation is shown in Figure 10.11.

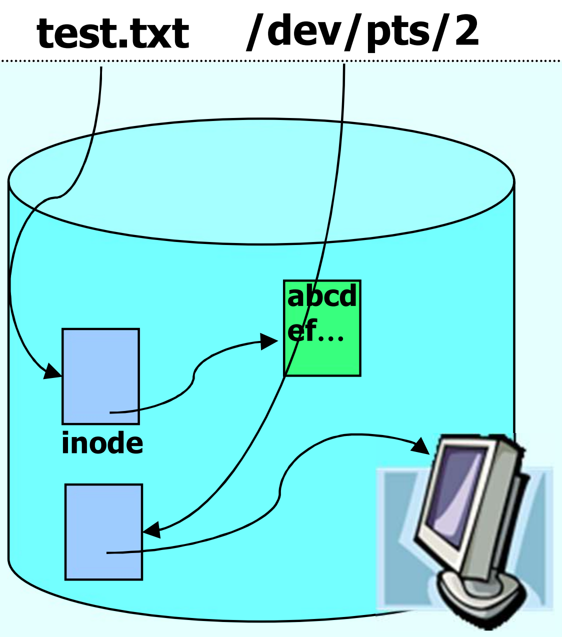


(Source: CSAPP)

File Programming: Advanced (3/6)

- Practice 6: device file

```
/* file_device.c, by choijm. choijm@dku.edu */
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define MAX_BUF 4
char fname[] = "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/pts/2", O_WRONLY);
        write(fd, buf, MAX_BUF);
        close(fd);
}
```



Devices such as terminal can be accessed using file interfaces

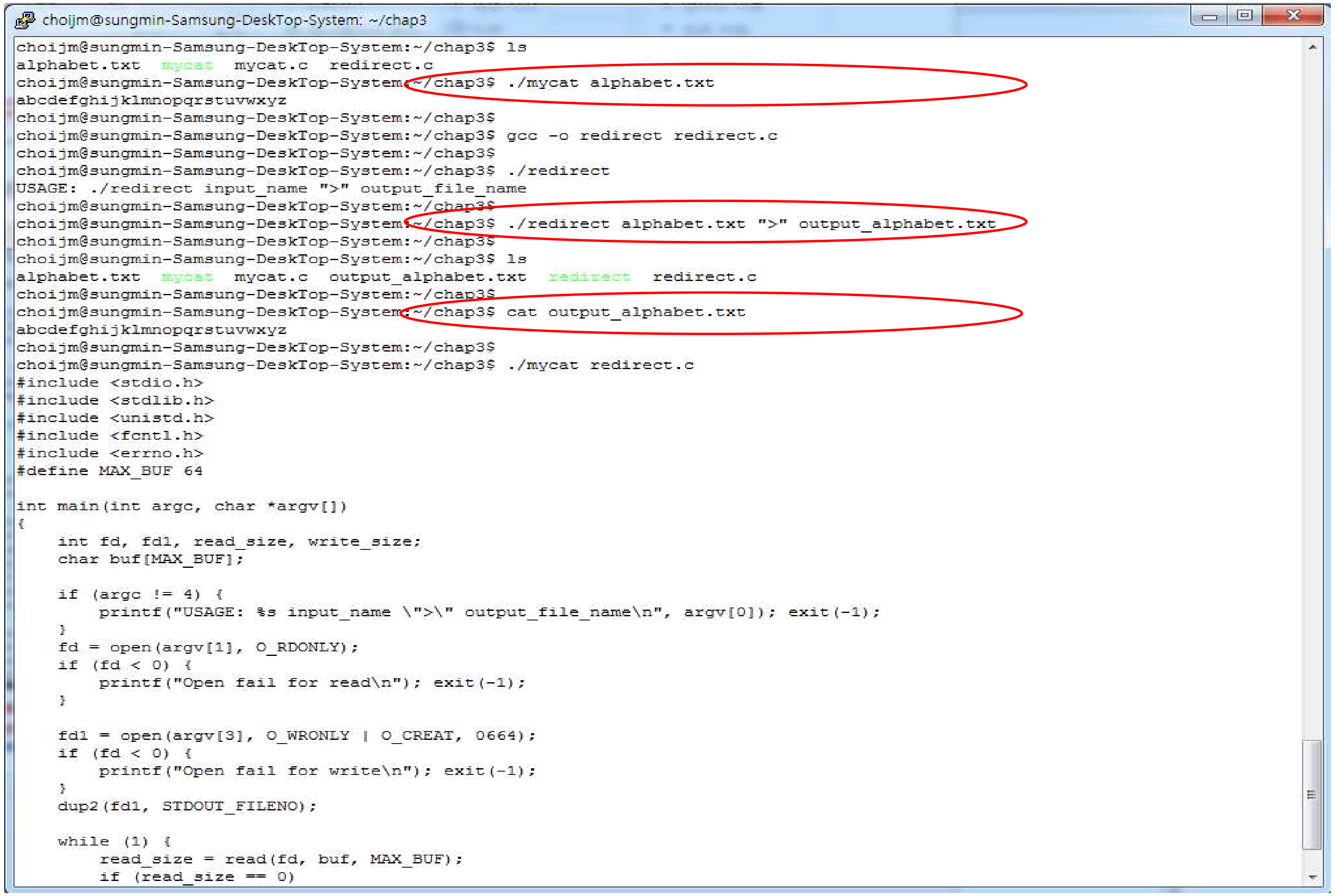
File Programming: Advanced (4/6)

- Practice 7: redirection (derived from “mycat” program)
  - ✓ Same fd but different objects

```
/* file_redirection.c, by choijm. choijm@dku.edu */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#define MAX_BUF 64

int main(int argc, char *argv[])
{
   int fd, fd1, read_size, write_size;
   char buf[MAX_BUF];

   if (argc != 4) {
       printf("USAGE: %s input_file_name ₩”>₩” output_file_name₩n", argv[0]);  exit(-1);
   }
   fd = open(argv[1], O_RDONLY);

   // for redirection. (eg. "mycat inputfile.txt > outputfile.txt“)
   // close(STDOUT_FILENO);
    fd1 = open(argv[3], O_RDWR | O_CREAT, 0641);
    dup2(fd1, STDOUT_FILENO);
    // redirection end

  while (1) {
      read_size = read(fd, buf, MAX_BUF);
      if (read_size == 0)
             break;
      write_size = write(STDOUT_FILENO, buf, read_size);
   }
   close(fd);
}
```

File Programming: Advanced (5/6)

- Practice 7: execution results



choijm@sungmin-Samsung-DeskTop-System: ~/chap3

```
choijm@sungmin-Samsung-DeskTop-System:~/chap3$ ls
alphabet.txt  mycat  mycat.c  redirect.c
choijm@sungmin-Samsung-DeskTop-System:~/chap3$ ./mycat alphabet.txt
abcdefghijklmnopqrstuvwxyz
choijm@sungmin-Samsung-DeskTop-System:~/chap3$
choijm@sungmin-Samsung-DeskTop-System:~/chap3$ gcc -o redirect redirect.c
choijm@sungmin-Samsung-DeskTop-System:~/chap3$
choijm@sungmin-Samsung-DeskTop-System:~/chap3$ ./redirect
USAGE: ./redirect input_name ">" output_file_name
choijm@sungmin-Samsung-DeskTop-System:~/chap3$
choijm@sungmin-Samsung-DeskTop-System:~/chap3$ ./redirect alphabet.txt ">" output_alphabet.txt
choijm@sungmin-Samsung-DeskTop-System:~/chap3$
choijm@sungmin-Samsung-DeskTop-System:~/chap3$ ls
alphabet.txt  mycat  mycat.c  output_alphabet.txt  redirect  redirect.c
choijm@sungmin-Samsung-DeskTop-System:~/chap3$
choijm@sungmin-Samsung-DeskTop-System:~/chap3$ cat output_alphabet.txt
abcdefghijklmnopqrstuvwxyz
choijm@sungmin-Samsung-DeskTop-System:~/chap3$
choijm@sungmin-Samsung-DeskTop-System:~/chap3$ ./mycat redirect.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#define MAX_BUF 64

int main(int argc, char *argv[])
{
    int fd, fd1, read_size, write_size;
    char buf[MAX_BUF];

    if (argc != 4) {
        printf("USAGE: %s input_name \">\" output_file_name\n", argv[0]); exit(-1);
    }
    fd = open(argv[1], O_RDONLY);
    if (fd < 0) {
        printf("Open fail for read\n"); exit(-1);
    }

    fd1 = open(argv[3], O_WRONLY | O_CREAT, 0664);
    if (fd < 0) {
        printf("Open fail for write\n"); exit(-1);
    }
    dup2(fd1, STDOUT_FILENO);

    while (1) {
        read_size = read(fd, buf, MAX_BUF);
        if (read_size == 0)
```

☞ **This is just an example. In general, redirection is in the form of "./redirection sourcefile.txt > outputfile.txt" (shell actually handle the redirection code)**

File Programming: Advanced (6/6)

- Discuss the tradeoff about the buffer size in read()/write()
  - ✓ Revisit mycat again: what if we change the MAX_BUF as **32** or **128**

```
/* mycat program, by choijm. choijm@dku.edu */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#define MAX_BUF 64

int main(int argc, char *argv[])
{
   int fd, read_size, write_size;
   char buf[MAX_BUF];

   if (argc != 2) {
      printf("USAGE: %s file_name\ n", argv[0]);  exit(-1);
   }
   fd = open(argv[1], O_RDONLY);
   if (fd < 0) {
      // open error handling
   }
   while (1) {
      read_size = read(fd, buf, MAX_BUF);
       if (read_size == 0)
              break;
      write_size = write(STDOUT_FILENO, buf, read_size);
   }
   close(fd);
}
```

Tracing system call

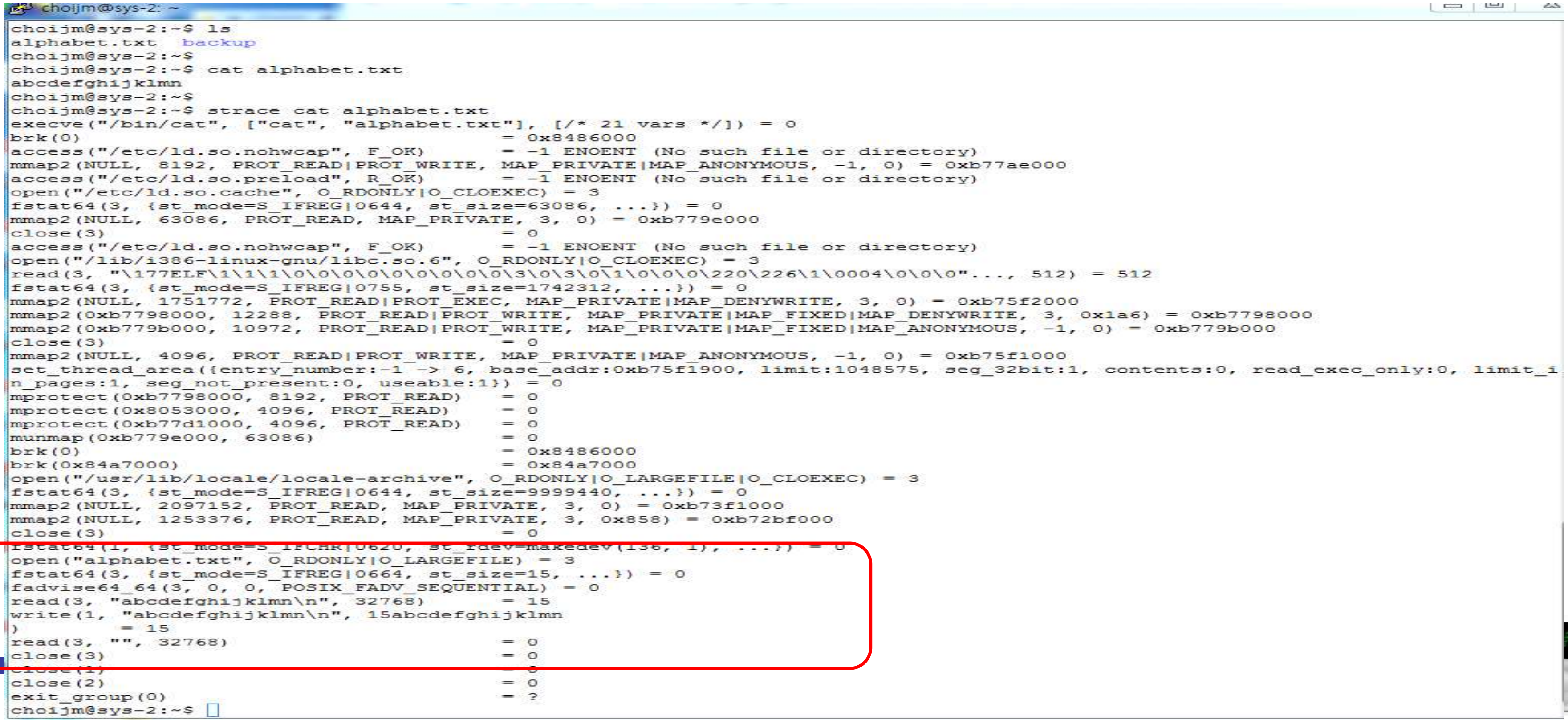
- Using “strace”

TIP: USE STRACE (AND SIMILAR TOOLS)

The `strace` tool provides an awesome way to see what programs are up to. By running it, you can trace which system calls a program makes, see the arguments and return codes, and generally get a very good idea of what is going on.

The tool also takes some arguments which can be quite useful. For example, `-f` follows any fork'd children too; `-t` reports the time of day at each call; `-e trace=open,close,read,write` only traces calls to those system calls and ignores all others. There are many more powerful flags — read the man pages and find out how to harness this wonderful tool.

(Source: Operating Systems: Three Easy Pieces)



choijm@sys-2: ~

```
choijm@sys-2:~$ ls
alphabet.txt  backup
choijm@sys-2:~$
choijm@sys-2:~$ cat alphabet.txt
abcdefghijklmn
choijm@sys-2:~$
choijm@sys-2:~$ strace cat alphabet.txt
execve("/bin/cat", ["cat", "alphabet.txt"], [/* 21 vars */]) = 0
brk(0)                                  = 0x8486000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb77ae000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=63086, ...}) = 0
mmap2(NULL, 63086, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb779e000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\220\226\1\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1742312, ...}) = 0
mmap2(NULL, 1751772, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb75f2000
mmap2(0xb7798000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a6) = 0xb7798000
mmap2(0xb779b000, 10972, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb779b000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb75f1000
set_thread_area({entry_number:-1 -> 6, base_addr:0xb75f1900, limit:1048575, seg_32bit:1, contents:0, read_exec_only:0, limit_i
n_pages:1, seg_not_present:0, useable:1}) = 0
mprotect(0xb7798000, 8192, PROT_READ)   = 0
mprotect(0x8053000, 4096, PROT_READ)    = 0
mprotect(0xb77d1000, 4096, PROT_READ)   = 0
munmap(0xb779e000, 63086)               = 0
brk(0)                                  = 0x8486000
brk(0x84a7000)                          = 0x84a7000
open("/usr/lib/locale/locale-archive", O_RDONLY|O_LARGEFILE|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=9999440, ...}) = 0
mmap2(NULL, 2097152, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb73f1000
mmap2(NULL, 1253376, PROT_READ, MAP_PRIVATE, 3, 0x858) = 0xb72bf000
close(3)                                = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...}) = 0
open("alphabet.txt", O_RDONLY|O_LARGEFILE) = 3
fstat64(3, {st_mode=S_IFREG|0664, st_size=15, ...}) = 0
fadvise64_64(3, 0, 0, POSIX_FADV_SEQUENTIAL) = 0
read(3, "abcdefghijklmn\n", 32768)      = 15
write(1, "abcdefghijklmn\n", 15abcdefghijklmn
)       = 15
read(3, "", 32768)                      = 0
close(3)                                = 0
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
choijm@sys-2:~$
```

Summary

- Understand the internal structure of disk
- Find out the relation between system programs for disk
  - ✓ Driver, file system, system call
- Grasp the role of the inode
- Make a program with file interfaces
  - ✓ open, read, write, close / lseek / device file and redirection

☞ **Homework 3: Make a command called "mycp"**

1.1 Requirements

- use argc and argv[ ]

- do not create a file if the same name already exists in current directory

- shows student's ID and date (using whoami and date)

1.2 Bonus: copy not only the contents but also the attributes

1.3 Write a report

- 1) Introduction: What to do, How, …

- 2) Design and Source code description

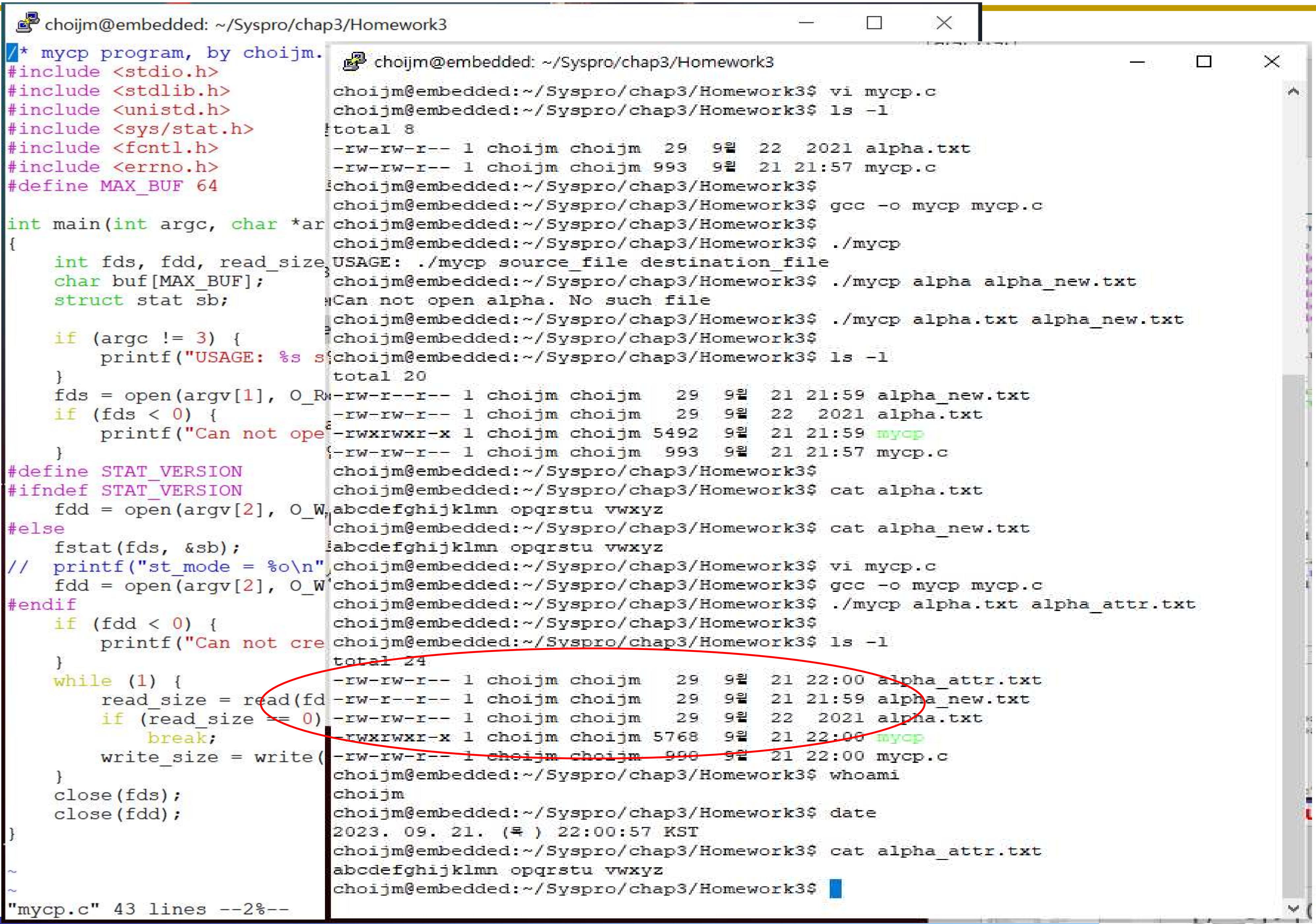
- 3) Execution Snapshots

- 4) Discussion: what you learn, issues, …

1.4 Deadline: Next week (same time)

1.5 How to submit? Send 1) report and 2) source code to TA using Google form (e.g. https://forms.gle/vBbJdJRQ3TUtZdTi9)

Homework 3: Snapshot example



choijm@embedded: ~/Syspro/chap3/Homework3

```
/* mycp program, by choijm.
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#define MAX_BUF 64

int main(int argc, char *ar
{
    int fds, fdd, read_size
    char buf[MAX_BUF];
    struct stat sb;

    if (argc != 3) {
        printf("USAGE: %s s
    }
    fds = open(argv[1], O_R
    if (fds < 0) {
        printf("Can not ope
    }
#define STAT_VERSION
#ifndef STAT_VERSION
    fdd = open(argv[2], O_W
#else
    fstat(fds, &sb);
//  printf("st_mode = %o\n"
    fdd = open(argv[2], O_W
#endif
    if (fdd < 0) {
        printf("Can not cre
    }
    while (1) {
        read_size = read(fd
        if (read_size == 0)
            break;
        write_size = write(
    }
    close(fds);
    close(fdd);
}
~
~
"mycp.c" 43 lines --2%--
```

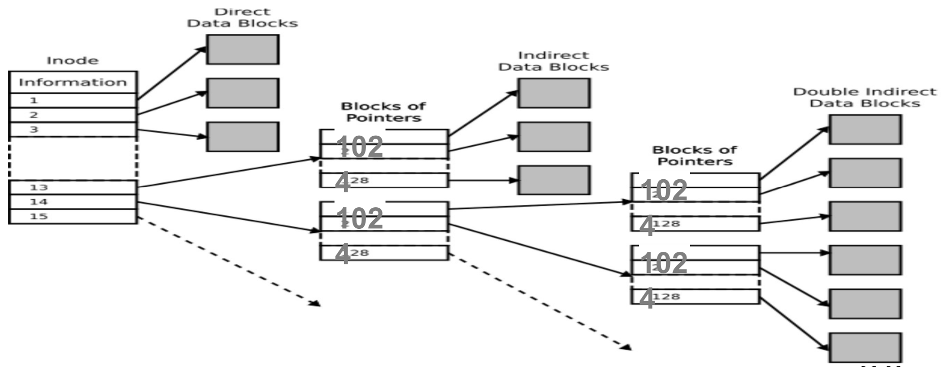
choijm@embedded: ~/Syspro/chap3/Homework3

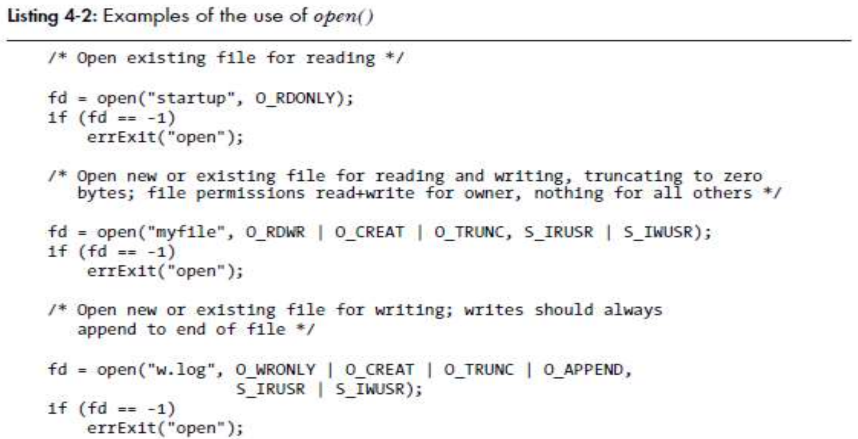
```
choijm@embedded:~/Syspro/chap3/Homework3$ vi mycp.c
choijm@embedded:~/Syspro/chap3/Homework3$ ls -l
total 8
-rw-rw-r-- 1 choijm choijm  29  9월  22  2021 alpha.txt
-rw-rw-r-- 1 choijm choijm 993  9월  21 21:57 mycp.c
choijm@embedded:~/Syspro/chap3/Homework3$
choijm@embedded:~/Syspro/chap3/Homework3$ gcc -o mycp mycp.c
choijm@embedded:~/Syspro/chap3/Homework3$
choijm@embedded:~/Syspro/chap3/Homework3$ ./mycp
USAGE: ./mycp source_file destination_file
choijm@embedded:~/Syspro/chap3/Homework3$ ./mycp alpha alpha_new.txt
Can not open alpha. No such file
choijm@embedded:~/Syspro/chap3/Homework3$ ./mycp alpha.txt alpha_new.txt
choijm@embedded:~/Syspro/chap3/Homework3$
choijm@embedded:~/Syspro/chap3/Homework3$ ls -l
total 20
-rw-r--r-- 1 choijm choijm   29  9월  21 21:59 alpha_new.txt
-rw-rw-r-- 1 choijm choijm   29  9월  22  2021 alpha.txt
-rwxrwxr-x 1 choijm choijm 5492  9월  21 21:59 mycp
-rw-rw-r-- 1 choijm choijm  993  9월  21 21:57 mycp.c
choijm@embedded:~/Syspro/chap3/Homework3$
choijm@embedded:~/Syspro/chap3/Homework3$ cat alpha.txt
abcdefghijklmn opqrstu vwxyz
choijm@embedded:~/Syspro/chap3/Homework3$ cat alpha_new.txt
abcdefghijklmn opqrstu vwxyz
choijm@embedded:~/Syspro/chap3/Homework3$ vi mycp.c
choijm@embedded:~/Syspro/chap3/Homework3$ gcc -o mycp mycp.c
choijm@embedded:~/Syspro/chap3/Homework3$ ./mycp alpha.txt alpha_attr.txt
choijm@embedded:~/Syspro/chap3/Homework3$
choijm@embedded:~/Syspro/chap3/Homework3$ ls -l
total 24
-rw-rw-r-- 1 choijm choijm   29  9월  21 22:00 alpha_attr.txt
-rw-r--r-- 1 choijm choijm   29  9월  21 21:59 alpha_new.txt
-rw-rw-r-- 1 choijm choijm   29  9월  22  2021 alpha.txt
-rwxrwxr-x 1 choijm choijm 5768  9월  21 22:00 mycp
-rw-rw-r-- 1 choijm choijm  998  9월  21 22:00 mycp.c
choijm@embedded:~/Syspro/chap3/Homework3$ whoami
choijm
choijm@embedded:~/Syspro/chap3/Homework3$ date
2023. 09. 21. (목) 22:00:57 KST
choijm@embedded:~/Syspro/chap3/Homework3$ cat alpha_attr.txt
abcdefghijklmn opqrstu vwxyz
choijm@embedded:~/Syspro/chap3/Homework3$
```

☞ **If you have any trouble to log in to the Linux server, Please inform your situation to TA.**

Quiz for this Lecture

- Quiz
  - ✓ 1. Describe the roles of three system programs for disk (using the term of abstraction).
  - ✓ 2. How large size can an inode support using direct block pointer? How about single, double, and triple indirect pointer?
  - ✓ 3. What is the functionality of O_NONBLOCK and O_SYNC in the flags of the open() system call?
  - ✓ 4. How can we figure out the size of a file using file interfaces that we learnt in this LN? (Note: 3 ways, NOT “ls –l”)
  - ✓ 5. SSD internally makes use of a SW called FTL (Flash Translation Layer). Discuss why SSD needs FTL based on the differences between Disk and Flash memory (2 key differences).



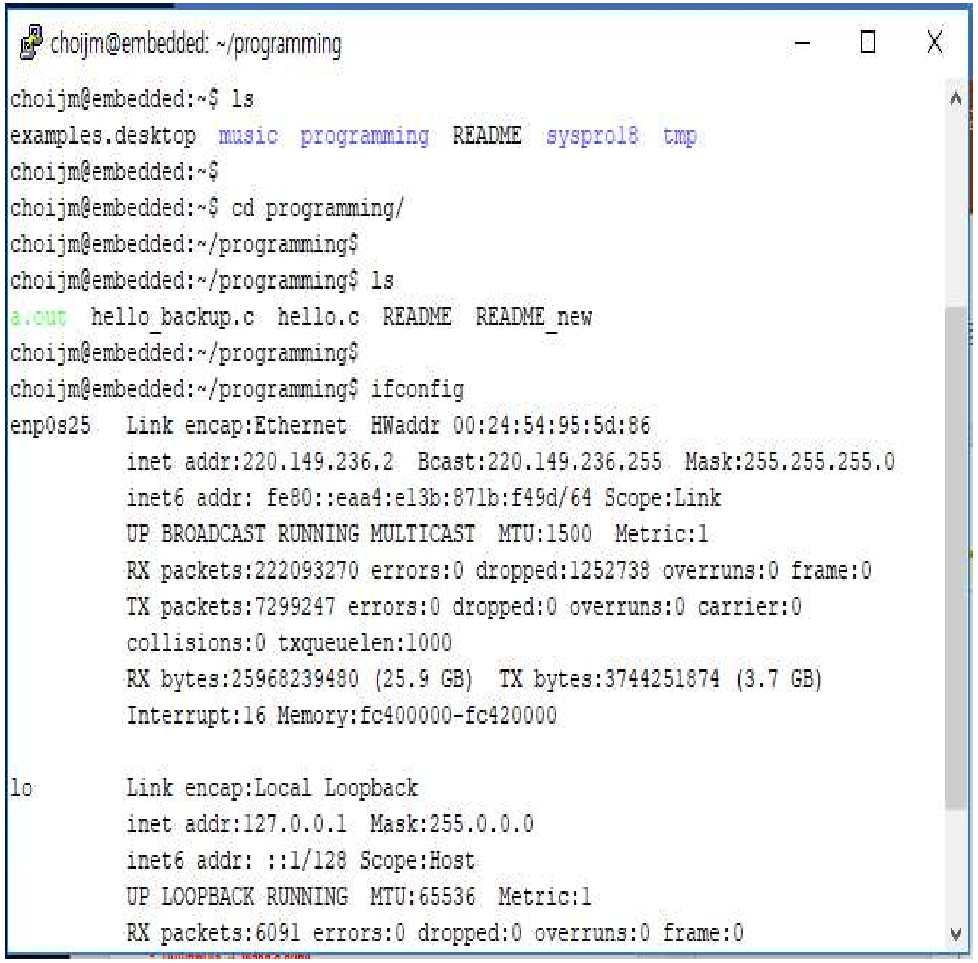


Listing 4-2: Examples of the use of *open()*

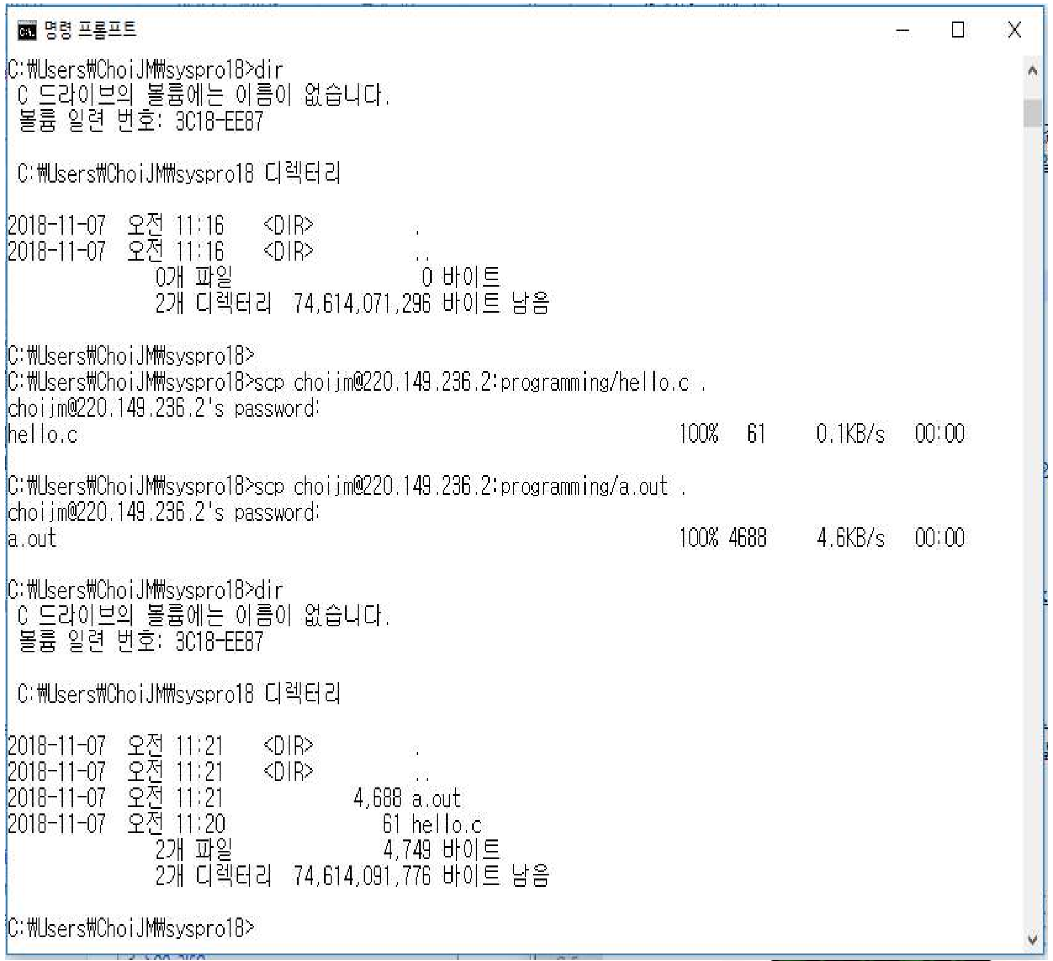
```
/* Open existing file for reading */

fd = open("startup", O_RDONLY);
if (fd == -1)
    errExit("open");

/* Open new or existing file for reading and writing, truncating to zero
   bytes; file permissions read+write for owner, nothing for all others */

fd = open("myfile", O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
if (fd == -1)
    errExit("open");

/* Open new or existing file for writing; writes should always
   append to end of file */

fd = open("w.log", O_WRONLY | O_CREAT | O_TRUNC | O_APPEND,
                   S_IRUSR | S_IWUSR);
if (fd == -1)
    errExit("open");
```

Appendix 1

- How to download files from Linux server?
  - ✓ scp (secure copy protocol)
    - A means of securely transferring computer files between a local host and a remote host or between two remote hosts



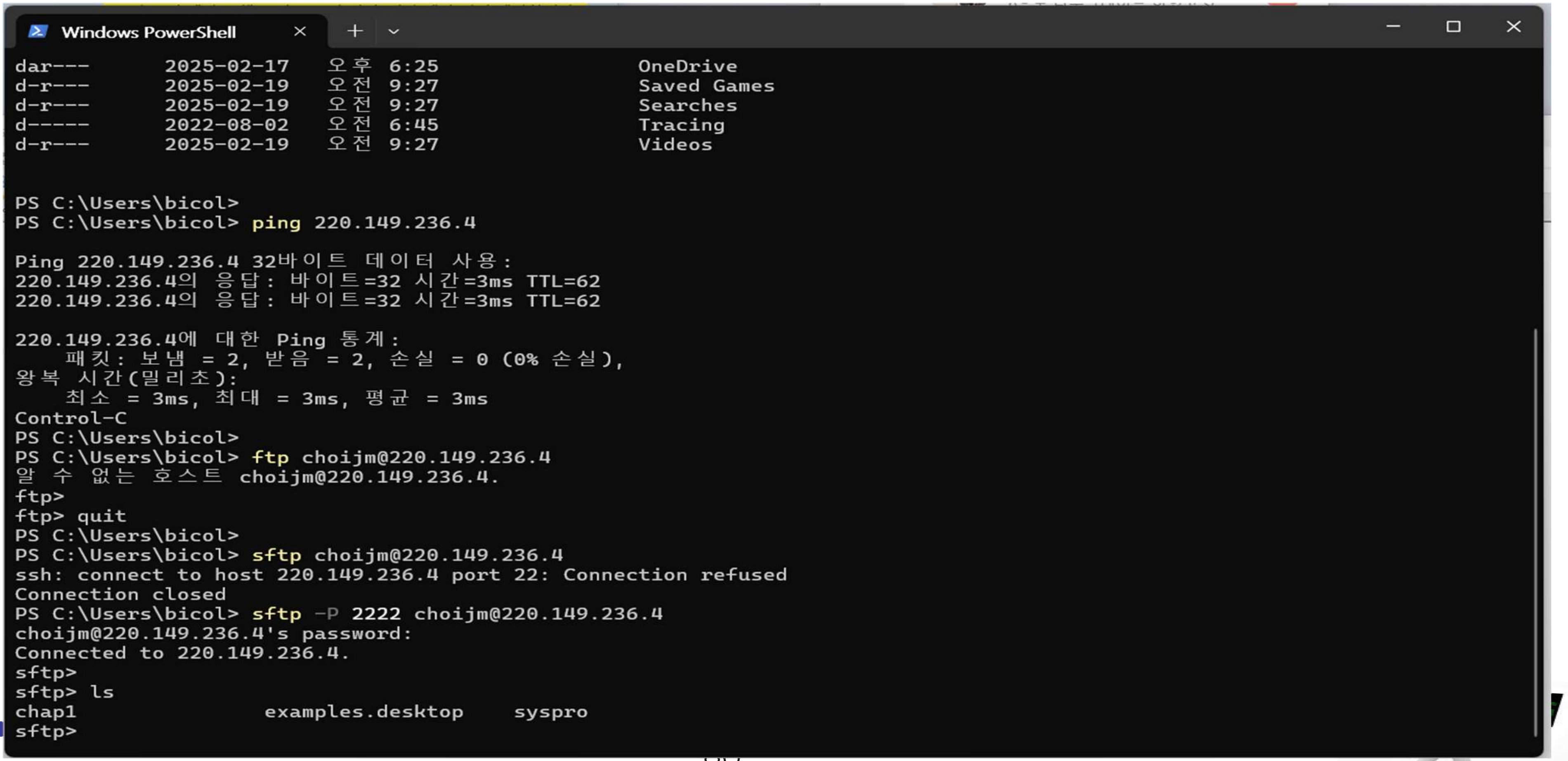
```
choijm@embedded:~$ ls
examples.desktop  music  programming  README  syspro18  tmp
choijm@embedded:~$
choijm@embedded:~$ cd programming/
choijm@embedded:~/programming$
choijm@embedded:~/programming$ ls
a.out  hello_backup.c  hello.c  README  README_new
choijm@embedded:~/programming$
choijm@embedded:~/programming$ ifconfig
enp0s25   Link encap:Ethernet  HWaddr 00:24:54:95:5d:86
          inet addr:220.149.236.2  Bcast:220.149.236.255  Mask:255.255.255.0
          inet6 addr: fe80::eaa4:e13b:871b:f49d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:222093270 errors:0 dropped:1252738 overruns:0 frame:0
          TX packets:7299247 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:25968239480 (25.9 GB)  TX bytes:3744251874 (3.7 GB)
          Interrupt:16 Memory:fc400000-fc420000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:6091 errors:0 dropped:0 overruns:0 frame:0
```



```
C:₩Users₩ChoiJM₩syspro18>dir
 C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: 3C18-EE87

 C:₩Users₩ChoiJM₩syspro18 디렉터리

2018-11-07  오전 11:16    <DIR>          .
2018-11-07  오전 11:16    <DIR>          ..
               0개 파일                   0 바이트
               2개 디렉터리  74,614,071,296 바이트 남음

C:₩Users₩ChoiJM₩syspro18>
C:₩Users₩ChoiJM₩syspro18>scp choijm@220.149.236.2:programming/hello.c .
choijm@220.149.236.2's password:
hello.c                                              100%   61     0.1KB/s   00:00

C:₩Users₩ChoiJM₩syspro18>scp choijm@220.149.236.2:programming/a.out .
choijm@220.149.236.2's password:
a.out                                                100% 4688     4.6KB/s   00:00

C:₩Users₩ChoiJM₩syspro18>dir
 C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: 3C18-EE87

 C:₩Users₩ChoiJM₩syspro18 디렉터리

2018-11-07  오전 11:21    <DIR>          .
2018-11-07  오전 11:21    <DIR>          ..
2018-11-07  오전 11:21             4,688 a.out
2018-11-07  오전 11:20                61 hello.c
               2개 파일               4,749 바이트
               2개 디렉터리  74,614,091,776 바이트 남음

C:₩Users₩ChoiJM₩syspro18>
```

Appendix 1

- How to download files from Linux server?
  - ✓ ftp (File Transfer Protocol)
    - a standard network protocol used for the transfer of computer files between a client and server on a computer network
  - ✓ sftp (secure ftp)

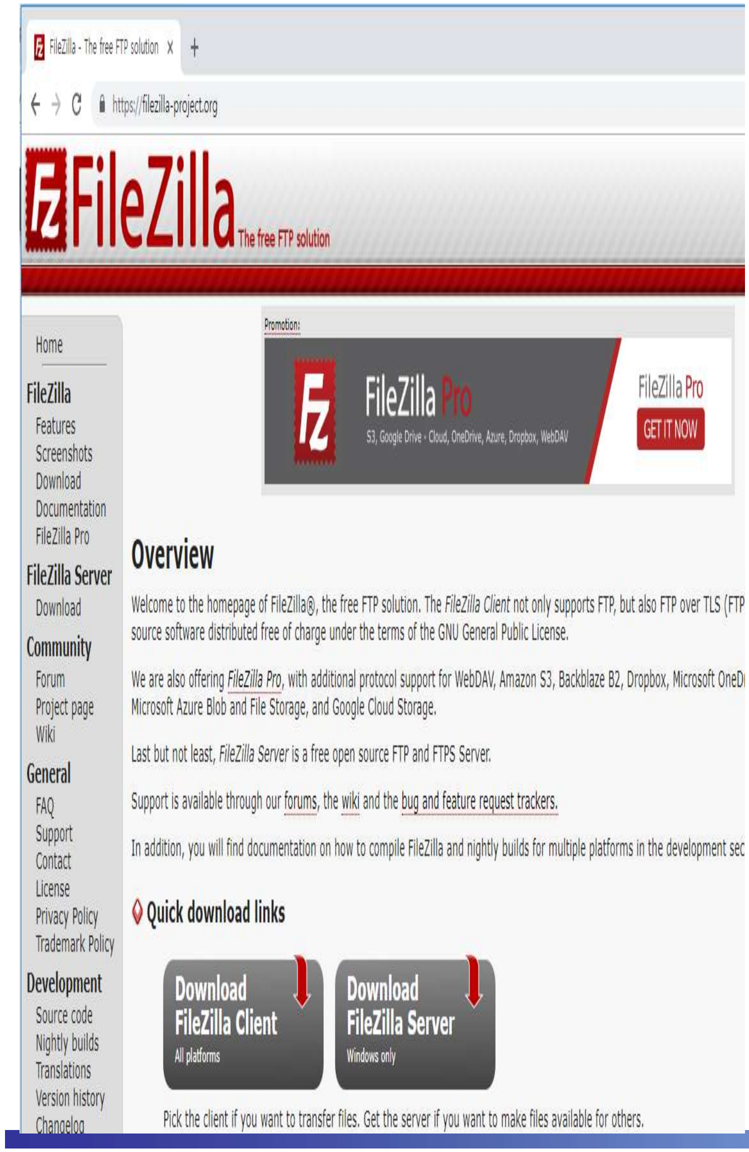


Windows PowerShell

```
dar---        2025-02-17   오후 6:25                OneDrive
d-r---        2025-02-19   오전 9:27                Saved Games
d-r---        2025-02-19   오전 9:27                Searches
d-----        2022-08-02   오전 6:45                Tracing
d-r---        2025-02-19   오전 9:27                Videos


PS C:\Users\bicol>
PS C:\Users\bicol> ping 220.149.236.4

Ping 220.149.236.4 32바이트 데이터 사용:
220.149.236.4의 응답: 바이트=32 시간=3ms TTL=62
220.149.236.4의 응답: 바이트=32 시간=3ms TTL=62

220.149.236.4에 대한 Ping 통계:
    패킷: 보냄 = 2, 받음 = 2, 손실 = 0 (0% 손실),
왕복 시간(밀리초):
    최소 = 3ms, 최대 = 3ms, 평균 = 3ms
Control-C
PS C:\Users\bicol>
PS C:\Users\bicol> ftp choijm@220.149.236.4
알 수 없는 호스트 choijm@220.149.236.4.
ftp>
ftp> quit
PS C:\Users\bicol>
PS C:\Users\bicol> sftp choijm@220.149.236.4
ssh: connect to host 220.149.236.4 port 22: Connection refused
Connection closed
PS C:\Users\bicol> sftp -P 2222 choijm@220.149.236.4
choijm@220.149.236.4's password:
Connected to 220.149.236.4.
sftp>
sftp> ls
chap1                examples.desktop    syspro
sftp>
```

Appendix 1

- How to download files from Linux server?
  - ✓ Using free ftp application with GUI (note: port = 2222)



FileZilla - The free FTP solution

https://filezilla-project.org

FileZilla The free FTP solution

Overview

Welcome to the homepage of FileZilla®, the free FTP solution. The *FileZilla Client* not only supports FTP, but also FTP over TLS (FTP
source software distributed free of charge under the terms of the GNU General Public License.

We are also offering *FileZilla Pro*, with additional protocol support for WebDAV, Amazon S3, Backblaze B2, Dropbox, Microsoft OneD
Microsoft Azure Blob and File Storage, and Google Cloud Storage.

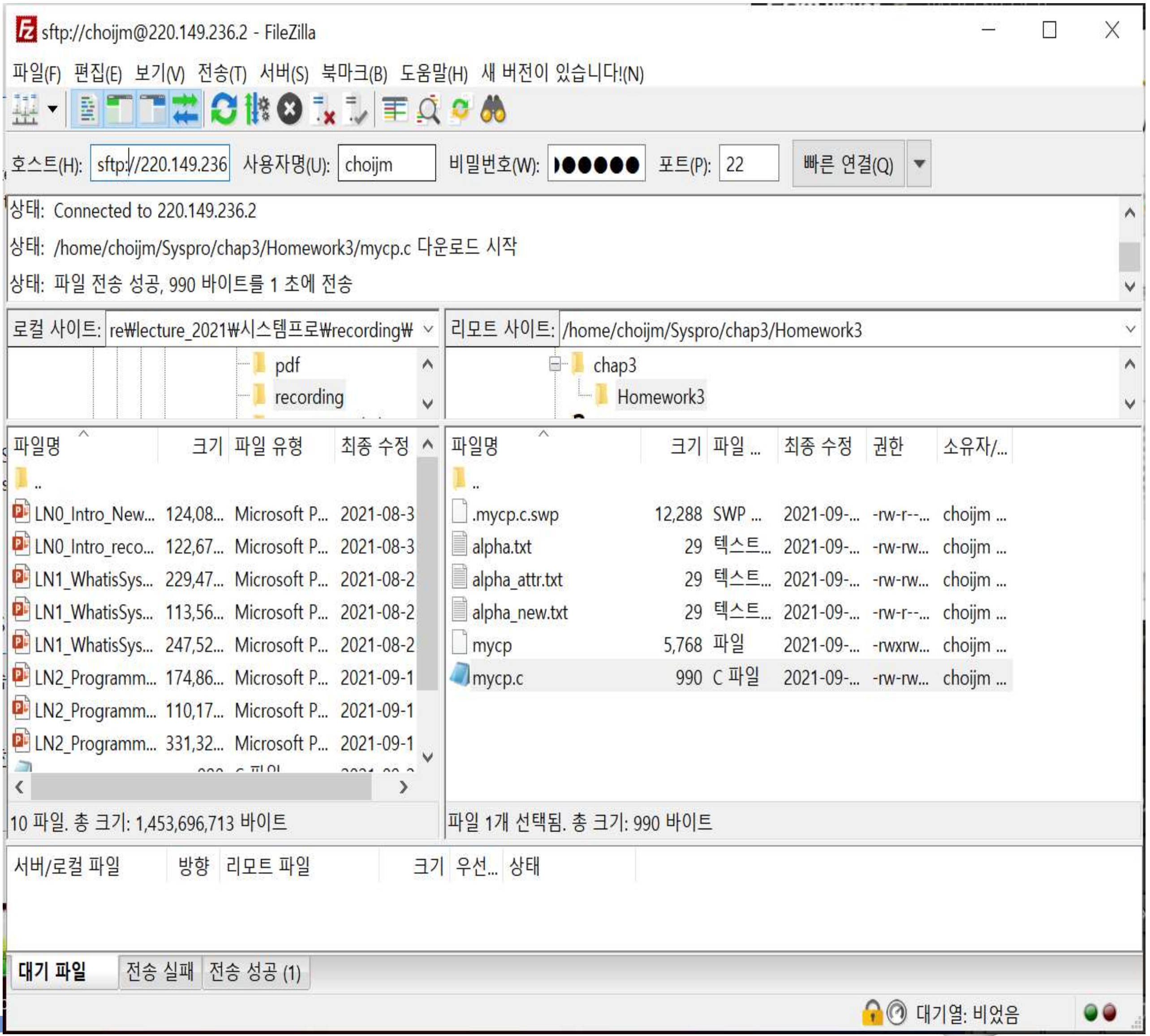
Last but not least, *FileZilla Server* is a free open source FTP and FTPS Server.

Support is available through our forums, the wiki and the bug and feature request trackers.

In addition, you will find documentation on how to compile FileZilla and nightly builds for multiple platforms in the development sec

Quick download links

Pick the client if you want to transfer files. Get the server if you want to make files available for others.



sftp://choijm@220.149.236.2 - FileZilla

파일(F) 편집(E) 보기(V) 전송(T) 서버(S) 북마크(B) 도움말(H) 새 버전이 있습니다!(N)
빠른 연결(Q)

호스트(H): sftp://220.149.236 사용자명(U): choijm 비밀번호(W): ●●●●●● 포트(P): 22

상태: Connected to 220.149.236.2
상태: /home/choijm/Syspro/chap3/Homework3/mycp.c 다운로드 시작
상태: 파일 전송 성공, 990 바이트를 1 초에 전송

로컬 사이트: re₩lecture_2021₩시스템프로₩recording₩

pdf
recording

리모트 사이트: /home/choijm/Syspro/chap3/Homework3

chap3
Homework3

파일명	크기	파일 유형	최종 수정
..			
LN0_Intro_New...	124,08...	Microsoft P...	2021-08-3
LN0_Intro_reco...	122,67...	Microsoft P...	2021-08-3
LN1_WhatisSys...	229,47...	Microsoft P...	2021-08-2
LN1_WhatisSys...	113,56...	Microsoft P...	2021-08-2
LN1_WhatisSys...	247,52...	Microsoft P...	2021-08-2
LN2_Programm...	174,86...	Microsoft P...	2021-09-1
LN2_Programm...	110,17...	Microsoft P...	2021-09-1
LN2_Programm...	331,32...	Microsoft P...	2021-09-1

파일명	크기	파일 ...	최종 수정	권한	소유자/...
..					
.mycp.c.swp	12,288	SWP ...	2021-09-...	-rw-r--...	choijm ...
alpha.txt	29	텍스트...	2021-09-...	-rw-rw...	choijm ...
alpha_attr.txt	29	텍스트...	2021-09-...	-rw-rw...	choijm ...
alpha_new.txt	29	텍스트...	2021-09-...	-rw-r--...	choijm ...
mycp	5,768	파일	2021-09-...	-rwxrw...	choijm ...
mycp.c	990	C 파일	2021-09-...	-rw-rw...	choijm ...

10 파일. 총 크기: 1,453,696,713 바이트

파일 1개 선택됨. 총 크기: 990 바이트

서버/로컬 파일	방향	리모트 파일	크기	우선...	상태

사사

- 본 교재는 2025년도 과학기술정보통신부 및 정보통신기획평가원의 ‘SW중심대학사업’ 지원을 받아 제작 되었습니다.
- 본 결과물의 내용을 전재할 수 없으며, 인용(재사용)할 때에는 반드시 과학기술정보통신부와 정보통신기획평가원이 지원한 ‘SW중심대학’의 결과물이라는 출처를 밝혀야 합니다.

IITP 정보통신기획평가원

디지털인재양성단 SW인재팀

SW중심대학이 무엇인가요?

SW중심대학은
대학교육을 SW중심으로 혁신함으로써,
SW전문인력을 양성하고
학생·기업·사회의 SW경쟁력을 강화해
진정한 SW가치 확산을 실현하는 대학을 말합니다.