File I/O

경희대학교 컴퓨터공학과

조 진 성

File in Linux

- Unified interface for all I/Os in UNIX
  - ✓ Regular(normal) files in file system
  - ✓ Special files for devices
    - terminal, keyboard, mouse, tape, hard disk, floppy disk, modem, etc.
    - /dev/*
  - ✓ Even a socket is similar to a file that is open

- System calls for I/O
  - ✓ **`open, read, write, close`**
  - ✓ **`lseek, fcntl, ioctl`**

- Predefined open file descriptors
  - ✓ **`#include <unistd.h>`**
  - ✓ **`STDIN_FILENO  (0)`**
  - ✓ **`STDOUT_FILENO (1)`**
  - ✓ **`STDERR_FILENO (2)`**

System Calls for File I/O

- Open a file or device
  - ✓ `#include <sys/types.h>`
  - ✓ `#include <sys/stat.h>`
  - ✓ `#include <fcntl.h>`
  - ✓ `int open(char *pathname, int oflag, ... /* , mode_t mode */);`
  - ✓ return: a file descriptor if OK, –1 on error
  - ✓ Options (the second parameter, **`oflag`**)
    - **`O_RDONLY, O_WRONLY, O_RDWR`**
    - **`O_APPEND, O_CREAT, O_EXCL, O_TRUNC, O_NONBLOCK, O_SYNC`**

- Close a file descriptor
  - ✓ `#include <unistd.h>`
  - ✓ `int close(int fd);`
  - ✓ return: 0 if OK, –1 on error

System Calls for File I/O (Cont'd)

- Read from a file descriptor
  - ✓ `#include <unistd.h>`
  - ✓ `ssize_t read(int fd, void *buf, size_t nbytes);`
  - ✓ return: number of bytes read if OK, 0 on end of file, –1 on error

- Write to a file descriptor
  - ✓ `#include <unistd.h>`
  - ✓ `ssize_t write(int fd, void *buf, size_t nbytes);`
  - ✓ return: number of bytes written if OK,–1 on error

System Calls for File I/O (Cont'd)

- Reposition read/write file offset
  - ✓ `#include <sys/types.h>`
  - ✓ `#include <unistd.h>`
  - ✓ `int lseek(int fd, off_t offset, int whence);`
  - ✓ return: new file offset if OK, –1 on error
  - ✓ The third parameter, **whence**
    - **SEEK_SET(0), SEEK_CUR(1), SEEK_END(2)**

- Create a new file or device
  - ✓ `#include <sys/types.h>`
  - ✓ `#include <sys/stat.h>`
  - ✓ `#include <fcntl.h>`
  - ✓ `int creat(char *pathname, mode_t mode);`
  - ✓ return: a file descriptor opened for write-only if OK, –1 on error
  - ✓ Equivalent to **`open(pathname, O_WRONLY | O_CREAT | O_TRUNC, mode);`**

System Calls for File I/O (Cont'd)

- Manipulate a file descriptor
  - ✓ `#include <sys/types.h>`
  - ✓ `#include <unistd.h>`
  - ✓ `#include <fcntl.h>`
  - ✓ `int fcntl(int fd, int cmd, ... /* int arg */ );`
  - ✓ return: depends on **`cmd`** if OK, 0 on end of file, –1 on error
  - ✓ Change properties of a file that is already open

- Control device
  - ✓ `#include <unistd.h>    /* SVR4 */`
  - ✓ `#include <sys/ioctl.h> /* BSD  */`
  - ✓ `int ioctl(int fd, int request, ...);`
  - ✓ return: depends on **`request`** if OK,–1 on error
  - ✓ Change I/O options for a given file descriptor
  - ✓ For terminal I/O, socket I/O, tape I/O, disk I/O, file I/O, etc.

Exercise

- Make my own **cp** program using file I/O system calls

```
$ gcc –o mycp mycp.c (or make mycp)
$ ./mycp mycp.c mycp.bak
$ ls –l mycp.c mycp.bak
```

- Make a big-hole file using **creat** & **lseek** system calls

```
$ gcc –o hole hole.c (or make hole)
$ ./hole
$ ls –l file.hole
```


Error Handling in System Calls and Libraries

- Global variable "**extern int errno**" in **<errno.h>**

```
#include <stdio.h>
#include <string.h>
#include <errno.h>

main(int argc, char *argv[])
{
  int fd;
  if ((fd = open("0123", O_RDONLY)) < 0)  {
    printf("%s: %s\n", argv[0], strerror(errno));
    perror(argv[0]);
  }
}
```


Streams and FILE Objects

- "Byte Stream"
- Logical structure for handling physical files with ease
- Predefined open streams
  - ✓ `#include <stdio.h>`
  - ✓ `FILE *stdin;`
  - ✓ `FILE *stdout;`
  - ✓ `FILE *stderr;`

Basic Four Steps

- Declaration of a pointer to FILE type

```
FILE *fp;
```

- Open a stream

```
if ((fp = fopen("/etc/passwd", "rt")) == NULL)  {
  printf("File open error\n");
  exit(1);
}
```

- Read or Write a stream

```
fputc('a', fp);
ch = fgetc(fp);
```

- Close a stream

```
fclose(fp);
```

Exercise

- List a text file with line numbers

```
$ gcc –o list list.c (or make list)
$ ./list list.c
```

Opening and Closing a Stream

- Open a stream
  - ✓ `#include <stdio.h>`
  - ✓ `FILE *fopen(char *pathname, char *type);`
  - ✓ return: a file pointer if OK, **`NULL`** on error
  - ✓ I/O type (the second parameter, **`type`**)
    - access mode: `r, w, a, r+, w+, a+`
    - file type: `t, b`

- Close a stream
  - ✓ `#include <stdio.h>`
  - ✓ `int fclose(FILE *fp);`
  - ✓ return: 0 if OK, **`EOF`** on error

Read and Write a Stream

- Character-at-a-time I/O
- Line-at-a-time I/O
- Direct I/O (Binary I/O)
- Formatted I/O

Character-At-A-Time I/O

- Input functions
  - ✓ `#include <stdio.h>`
  - ✓ `int getc(FILE *fp);`
  - ✓ `int fgetc(FILE *fp);`
  - ✓ all return: next character if OK, **EOF** on end of file or error

- Output functions
  - ✓ `#include <stdio.h>`
  - ✓ `int putc(int ch, FILE *fp);`
  - ✓ `int fputc(int ch, FILE *fp);`
  - ✓ all return: **ch** if OK, **EOF** on error

Line-At-A-Time I/O

- Input functions
  - ✓ `#include <stdio.h>`
  - ✓ `char *fgets(char *buf, int max, FILE *fp);`
  - ✓ return: **`buf`** if OK, **`NULL`** on end of file or error

- Output functions
  - ✓ `#include <stdio.h>`
  - ✓ `int fputs(char *str, FILE *fp);`
  - ✓ return: non-negative value if OK, **`EOF`** on error

Direct I/O (Binary I/O)

- Input functions
  - ✓ `#include <stdio.h>`
  - ✓ `size_t fread(void *ptr, size_t size, size_t nobj, FILE *fp);`
  - ✓ return: number of objects read if OK, 0 on end of file or error
- Output functions
  - ✓ `#include <stdio.h>`
  - ✓ `size_t fwrite(void *ptr, size_t size, size_t nobj, FILE *fp);`
  - ✓ return: number of objects written if OK, **0** on end of file or error
- Examples

```
struct rec tmp;
struct rec item[10];
fread(&tmp, sizeof(struct rec), 1, fp);
fwrite(item, sizeof(struct rec), 10, fp);
```


Formatted I/O

- Input functions
  - ✓ `#include <stdio.h>`
  - ✓ `int fscanf(FILE *fp, char *format, ...);`
  - ✓ `int sscanf(char *buf, char *format, ...);`
  - ✓ return: number of items assigned if OK, **EOF** on end of file or error

- Output functions
  - ✓ `#include <stdio.h>`
  - ✓ `int fprintf(FILE *fp, char *format, ...);`
  - ✓ `int sprintf(char *buf, char *format, ...);`
  - ✓ return: number of characters written if OK, negative value on error

Positioning a Stream

- Functions
  - ✓ `#include <stdio.h>`
  - ✓ `#include <unistd.h>`
  - ✓ `int fseek(FILE *fp, long offset, int whence);`
  - ✓ return: 0 if OK, nonzero on error
  - ✓ The third parameter, **whence**
    - **SEEK_SET(0), SEEK_CUR(1), SEEK_END(2)**

  - ✓ `#include <stdio.h>`
  - ✓ `long ftell(FILE *fp);`
  - ✓ return: current file position if OK, -1L on error

  - ✓ `#include <stdio.h>`
  - ✓ `void rewind(FILE *fp);`
  - ✓ set file position to the beginning of the file

Miscellaneous

- Functions
  - ✓ `#include <stdio.h>`
  - ✓ `int ungetc(int ch, FILE *fp);`
  - ✓ return: **ch** if OK, **EOF** on error

  - ✓ `#include <stdio.h>`
  - ✓ `int ferror(FILE *fp);`
  - ✓ `int feof(FILE *fp);`
  - ✓ return: nonzero if true, 0 otherwise

  - ✓ `#include <stdio.h>`
  - ✓ `int fflush(FILE *fp);`
  - ✓ return: 0 if OK, **EOF** on error
  - ✓ Flush buffer cache, i.e., write-back to the disk

Error Handling in File I/O

- Error occurrence
  - ✓ **`fopen`** for read → frequently
    - Must check it
  - ✓ **`fopen`** for write → infrequently
    - Don't have to check it, but should check it for consistency
  - ✓ read functions → infrequently
    - But, must check EOF
  - ✓ write functions → infrequently
    - Don't have to check it
  - ✓ **`fclose`** → infrequently
    - Don't have to check it

Exercise

- Make a text-copy program using character-at-a-time I/O stream

```
$ gcc –o tcp1 tcp1.c (or make tcp1)
$ ./tcp1 tcp1.c tcp1.bak
$ ls –l tcp1.c tcp1.bak
```

- Make a text-copy program using line-at-a-time I/O stream

```
$ gcc –o tcp2 tcp2.c (or make tcp2)
$ ./tcp2 tcp2.c tcp2.bak
$ ls –l tcp2.c tcp2.bak
```

- Make my own **cp** program using binary I/O stream

```
$ gcc –o mycp2 mycp2.c (or make mycp2)
$ ./mycp2 mycp2.c mycp2.bak
$ ls –l mycp2.c mycp2.bak
```


Exercise (Cont'd)

- Split a file into two & Merge two files into one using binary I/O stream

```
$ gcc –o split split.c (or make split)
$ gcc –o merge merge.c (or make merge)
$ ./split merge a b
$ ls –l merge a b
$ ./merge a b merge.new
$ chmod a+x merge.new
$ ./merge.new
```


Exercise (Cont'd)

- Another binary I/O example
  - ✓ Type-in a data file in text format
  - ✓ Convert a text file into a binary file
  - ✓ Access the binary file for search, insert, modify, etc.

```
$ gcc –o conv conv.c (or make conv)
$ gcc –o access access.c (or make access)
$ ./conv test.in test.out
$ ls –l test.out
$ ./access test.out
```

- What does this program do?

```
$ gcc –o tab tab.c (or make tab)
$ ./tab *.c
```

Summary

- System calls in Linux for file I/O
  - ✓ **open, close, read, write**
  - ✓ **lseek, create, fcntl, ioctl**

- C libraries for file I/O
  - ✓ **fopen, fclose, fread, fwrite**
  - ✓ **fgetc, fputc, fgets, fputs**
  - ✓ **fscanf, fprintf, sscanf, sprintf**
  - ✓ **fseek, ftell, rewind**
  - ✓ **ungetc, ferror, feof, fflush**