Contents

1. Preface/Introduction
2. Standardization and Implementation
3. File I/O
4. Standard I/O Library
5. Files and Directories
6. System Data Files and Information
7. Environment of a Unix Process
8. Process Control
9. Signals
10. Inter-process Communication

Standard I/O Library

- A major revision by Dennis Ritchie in 1975 based on the Portable I/O library by Mike Lesk
- An ANSI C standard
  - Easy to use and portable
  - Details handled:
    - Buffer allocation, optimal-sized I/O chunks, better interface, etc.

Standard I/O Library

- Difference from File I/O
  - File Pointers vs File Descriptors
  - fopen vs open
    - When a file is opened/created, a *stream* is associated with the file.
    - FILE object
      - File descriptor, buffer size, # of remaining chars, an error flag, and the like.
  - stdin, sdtout, stderr defined in <stdio.h>
    - STDIO_FILENO, STDOUT_FILENO,…

Buffering

- Goal
  - Use the minimum number of read and write calls.
- Types
  - Fully Buffered
    - Actual I/O occurs when the buffer is filled up.
    - A buffer is automatically allocated when the first-time I/O is performed on a stream.
    - flush: standard I/O lib vs terminal driver

Buffering

- Line Buffered
  - Perform I/O when a newline char is encountered! – usually for terminals.
  - Caveats
    - The filling of a fixed buffer could trigger I/O.
    - The flushing of all line-buffered outputs if input is requested.
- Unbuffered
  - Expect to output asap, e.g. using write()
  - E.g., stderr

Buffering

- ANSI C Requirements
  - Fully buffered for stdin and stdout unless interactive devices are referred to.
    - SVR4/4.3+BSD – line buffered
  - Standard error is never fully buffered.

```
#include <stdio.h>
int fflush(FILE *fp);
```

- All output streams are flushed if fp == NULL

Buffering

#include <stdio.h>
void setbuf(FILE *fp, char *buf);
void setvbuf(FILE *fp, char *buf, int mode,
size_t size);

- Full/line buffering if buf is not NULL (BUFSIZ)
  - Terminals
- mode: _IOFBF, IOLBF, _IONBF (<stdio.h>)
  - Optional size → st_blksize (stat())
- #define BUFSIZ 1024 (<stdio.h>)
- They must be called before any op is performed on the streams!

Buffering

- Possible Memory Access Errors
  - Use automatic allocation – NULL for *buf in setvbuf() – bookkeeping

	mode	buf	len	type
setbuf		non-null	BUFSIZ	FB/LB
		NULL		NB
setvbuf	FB	non-null	any size	FB
	FB	NULL	st_blksize	FB
	LB	non-null	any size	LB
	LB	NULL	st_blksize	LB
	NB	ignored	no buffered	NB

Standard I/O Library - Open

#include <stdio.h>

FILE *fopen(const char *pathname, const char *type);

FILE *freopen(const char *pathname, const char *type, FILE *fp);

- fopen/freopen opens a specified file! – POSIX.1
  - Close *fp* stream first!
- New files created by a or w have r/w rights for all

Type	r	w	a	r+	w+	a+
File exists?	Y			Y		
Truncate		Y			Y	
R	Y			Y	Y	Y
W		Y	Y	Y	Y	Y
W only at end			Y			Y

Standard I/O Library - Open

#include <stdio.h>

FILE *fdopen(int fildes, const char *type);

- Associate a standard I/O stream with an existing file descriptor – POSIX.1
  - Pipes, network channels
  - No truncating for the file for “w”
- b (in rb, wb, ab, r+b, …) stands for a binary file – no effect for Unix kernel
- O_APPEND supports multiple access.
- Interleaved R&W restrictions – intervening fflush (WR), fseek(WR/RW), fsetpos (WR/RW), rewind (WR/RW), EOF (RW)

Standard I/O Library – Open/Close

#include <stdio.h>

int fclose(FILE *fp);

- Flush buffered output
- Discard buffered input
- All I/O streams are closed after the process exits.

- setbuf or setvbuf to change the buffering of a file before any operation on the stream.

Standard I/O Library – Reading/Writing

- Unformatted I/O
  - Character-at-a-time I/O, e.g., getc
    - Buffering handled by standard I/O lib
  - Line-at-a-time I/O, e.g., fgets
    - Buffer limit might need to be specified.
  - Direct I/O, e.g., fread
    - Read/write a number of objects of a specified size.
    - An ANSI C term, e.g., = object-at-a-time I/O

Standard I/O Library – Reading/Writing

```
#include <stdio.h>
int getc(FILE *fp);
int fgetc(FILE *fp);
int getchar(void);
```

- getchar == getc(stdin)
- Differences between getc and fgetc
  - getc could be a macro
  - Argument's side effect, exec time, passing of the function address.
- unsigned char converted to int in returning

Standard I/O Library – Reading/Writing

```
#include <stdio.h>
int ferror(FILE *fp);
int feof(FILE *fp);
void clearerr(FILE *fp);
int ungetc(int c, FILE *fp);
```

- An error flag and an EOF flag for each FILE
- No pushing back of EOF (i.e., -1)
  - No need to be the same char read!

Standard I/O Library – Reading/Writing

#include <stdio.h>

int putc(int c, FILE *fp);

int fputc(int c, FILE *fp);

int putchar(int c);

- putchar(c) == putc(c, stdout)
- Differences between putc and fputc
  - putc() can be a macro.

Line-at-a-Time I/O

#include <stdio.h>

char *fgets(char *buf, int n, FILE *fp);

- Include '\n' and be terminated by *null*
- Could return a partial line if the line is too long.

char *gets(char *buf);

- Read from stdin.
- No buffer size is specified → overflow
- *buf does not include '\n' and is terminated by *null.*

Line-at-a-Time I/O

#include <stdio.h>

char *fputs(const char *str, FILE *fp);

- Include '\n' and be terminated by *null.*
- No need for line-at-a-time output.

char *puts(const char *str);

- *str does not include '\n' and is terminated by *null.*
- puts then writes '\n' to stdout.

Standard I/O Efficiency

- Program 5.1 – Page 131
  - Copy stdin to stdout: getc → putc
- Program 5.2 – Page 132
  - Copy stdin to stdout: fgets → fputs

Function	Usr CPU	Sys CPU	Clock	Program	
Fig3.1	0.0s	0.3s	0.3s		
fgets, fputs	2.2s	0.3s	2.6s	184B	text
getc, putc	4.3s	0.3s	4.8s	384B	
fgetc, fputc	4.6s	0.3s	5.0s	152B	spac
1B/Fig3.1	23.8s	397.9s	423.4s		

Loops in char/line-at-a-time cycles!

the same # of kernel calls!

Binary I/O

- Objectives
  - Read/write a structure at a time, which could contains null or '\n'.

```
#include <stdio.h>
size_t fread(void *ptr, size_t size, size_t nobj,
   FILE *fp);
size_t fwrite(const void *ptr, size_t size, size_t
   nobj, FILE *fp);
```

  - Reads less than the specified number of objects → error or EOF → ferror, feof
  - Write error if less than the specified number of objects are written.

Binary I/O

- Example 1

```
float data[10];
if (fwrite(&data[2], sizeof(float), 4, fp) != 4)
    err_sys("fwrite error");
```

- Example 2

```
struct {
        short count;
        long total;
        char name[NAMESIZE];
} item;
if (fwrite(&item, sizeof(item), 1, fp) != 1)
    err_sys("fwrite error");
```

Binary I/O

- Not portable for programs using fread and fwrite
  1. The offset of a member in a structure can differ between compilers and systems (due to alignment).
  2. The binary formats for various data types, such as integers, could be different over different machines.

Positioning-a-Stream

```
#include <stdio.h>
long ftell(FILE *fp);
int fseek(FILE *fp, long offset, int whence);
void rewind(FILE *fp);
```

- Assumption: a file's position can be stored in a long (since Version 7)
- whence: same as lseek
  - Binary files: No requirements for SEEK_END under ANSI C (good under Unix, possible padding for other systems).
  - Text files: SEEK_SET only – 0 or returned value by ftell (different formats for some sys).

Positioning-a-Stream

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

- ANSI C standard
- Good for non-Unix systems
- A new data type fpos_t

Formatted I/O – Output

```
#include <stdio.h>
int printf(const char *format, …);
int fprintf(FILE *fp, const char *format, …);
int sprintf(char *buf, const char *format, …);
```

- Overflow is possible for sprintf() – '\0' appended at the end of the string.

```
int vprintf(const char *format, var_list arg);
int vfprintf(FILE *fp, const char *format, var_list
  arg);
int vsprintf(char *buf, const char *format,
  var_list arg);
```

Formatted I/O – Input

```
#include <stdio.h>
int scanf(const char *format, …);
int fscanf(FILE *fp, const char
  *format, …);
int sscanf(char *buf, const char
  *format, …);
```


Implementation Details

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

- Get filedes for fcntl, dup, etc
- See <stdio.h> for per-stream flags, etc.

- Program 5.3 – Page 139
  - Printing buffering for various I/O streams
  - stdin, stdout – line-buffered, buf size
  - stderr – unbuffered, buf size
  - files: fully-buffered, buf size

Temporary Files

#include <stdio.h>

char *tmpnam(char *ptr);

- TMP_MAX in <stdio.h> /* = 25, ANSI C */
- If ptr == null, the pointer to the pathname is returned (L_tmpnam # of bytes assumed if ptr != null).

FILE *tmpfile(void);

- wb+ an empty binary file.
- Unlink the file immediately after it is created!

- Program 5.4 – Page 141
  - tmpnam and tmpfile

Temporary Files

#include <stdio.h>

char *tempnam(const char *directory, const char *prefix);

- TMPDIR
- *directory is null?
- P_tmpdir in <stdio.h>
- /tmp
- /* prefix could be up to 5 chars */
- Not POSIX.1 and ANSI C, but XPG3 (SVR4, 4.3+BSD)

- Program 5.5 – Page 142
  - tempnam

Alternatives to Standard I/O

- Main Issue
  - Too many data copyings
    - kernel → standard I/O buffer
    - standard I/O buffer → our buffer
- Alternatives
  - Fast I/O Library (fio) – pointer
  - sfio
    - Represent files/memory regions under I/O streams, and stack processing modules above I/O streams.
  - mmap