# C I/O

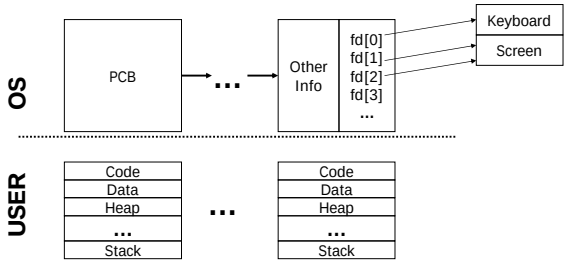
Stanley Yao
Computer Science Department
University of Arizona

## Stream

- A logical model to simplify and unify the complicated I/O behaviors
- Analogy: unify various device and resource with the same interface – files
- Stream
  - Stream is a sequence of lines
  - A line is a sequence of characters ending with newline



## File






## I/O Levels

| | |
|---|---|
| C stdio Library (Formatted) | FILE * wrapper, printf(), fprintf(), scan(), ... |
| C stdio Library (Standard) | FILE * wrapper, fopen(), fread(), fwrite(), ... **Buffered I/O** |
| UNIX System Call C API | File descriptor, open(), read(), write(), ... **Unbuffered I/O** |
| UNIX System Service Routines | |



## C Standard I/O

- int getchar(void);
- int putchar(int c);

- char *gets(char *str);
- int puts(const char *str);



## File I/O

- FILE *fopen(const char *filename, const char *mode);
  - Binary stream
  - Text stream
    - newline ⇔ carriage return (CR) + linefeed (LF)
- int fclose(FILE *stream);

- int feof(FILE *stream);
- int fflush(FILE *stream);

## File I/O (cont.)

- size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
- size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
- int fseek(FILE *stream, long int offset, int whence);



## File I/O (cont.)

- int getc(FILE *stream);
- int putc(int c, FILE *stream);
- char *fgets(char *str, int n, FILE *stream);
- int fputs(const char *str, FILE *stream);



## Formatted I/O

- int fprintf(FILE **stream*, const char **format*, ...);
- int printf(const char **format*, ...);
- int sprintf(char **str*, const char **format*, ...);

- int fscanf(FILE **stream*, const char **format*, ...);
- int scanf(const char **format*, ...);
- int sscanf(const char **str*, const char **format*, ...);



## Format String for printf()

- Normal characters: “abc”
- Special characters: “abc\n”
  - \n -- newline
  - \t -- tab
  - \r -- carriage return
  - \\ -- backslash character
  - \” - double-quote
  - \’ - single-quote
- Conversion specification: “abc %d\n”



## Conversion Specification

- %[flags][width][.precision][modifier][type]
- Flags: -, +, space, 0, #
- Width: minimum field width
- Precision
  - Integer: minimum digits to printed (padded with 0 when necessary)
  - Float: number of digits after the decimal point
  - Float (g, G): number of significant digits to print
  - String: max number of characters to display
- Modifier
  - h: short (for integer)
  - l: long (for integer)
  - L: long double (for float)



## Conversion Specification (cont.)

- Conversion Type
  - %d -- signed decimal integer
  - %u -- unsigned decimal integer
  - %x -- hexadecimal integer (characters in lower-case)
  - %X -- hexadecimal integer (characters in upper-case)
  - %s -- null-terminated string
  - %f -- floating-point number
  - %p -- pointer
  - %% -- percent sign (doesn’t consume parameter)

## Varying Width or Precision

- *: value is specified by the next argument
- printf() returns the number of characters printed.

- printf("%.*s", 3, "hello world");
  - Output: hel
  - Return: 4
- printf("%*.*f", 20, 2, 1234.56789);
  - Output: 1234.57
  - Return: 21



## Format String for scanf()

- Conversion Specifier: input field, stored in the corresponding argument
- Other characters: must be matched from the input, but are not stored in arguments

- If the input does not match then the function stops scanning and returns.
- Whitespace matches any amount of whitespaces in the input, including none.
- [ ] matches a sequence of characters from those between the brackets.



## scanf() Return Value

- Successful: number of successfully matched and assigned input items
- Input ends before the 1st match: EOF
- Read error: EOF (errno set)



## strtok()

- **char *strtok(char *s, char *delim)**
- Break a string into tokens by multiple calls to strtok().
- Original string **s** is modified ('\0' is inserted)
- Return pointers pointing to the next token



## System Error Message

- Direct previous system or library function call

- int errno;
- void perror(const char *s);
- char *strerror(int errnum);



## Acknowledgement


- John H. Hartman, *Classnotes for Csc352-Spring03*, CS Dept., University of Arizona, 2003
- Brian W. Kernighan, Dennis M. Ritchie, *The C Programming Language (2nd Ed.),* Prentice Hall, 1988