System and Device Programming

System Input/Output

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

The C Standard Library

- The standard C I/O library is specified by the ISO C standard
  - It was written by Dennis Ritchie around 1975
    - Surprisingly little has changed since then
  - It has been implemented on many operating systems other than the UNIX System
  - It handles several details, such as buffer allocation and performing I/O in optimal-sized chunks
  - It is easy to use

The C Standard Library

- I/O operations in C can be performed through different categories of functions
  - Streams and files
    - fopen, fclose
  - Character by character
    - getc, fgetc, putc, fputc
  - Row by row
    - gets, fgets, puts, fputs
  - Formatted I/O
    - scanf, fscanf, printf, fprint
  - Binary I/O
    - fread, fwrite

The C Standard Library

Prototypes

```
#include <stdio.h>

FILE *fopen (char *path, char *type);
FILE *fclose (FILE *fp);
int getc (FILE *fp);
int fgetc (FILE *fp);
int putc (int c, FILE *fp);
int fputc (int c, FILE *fp);
char gets (char *buf);
char *fgets (char *buf, int n, FILE *fp);
int puts (char *buf);
int *fputs (char *buf, FILE *fp);
int scanf (char format, …);
int fscanf (FILE *fp, char format, …);
int printf (char format, …);
int fprintf (FILE *fp, char format, …);

size_t fread (void *ptr, size_t size, size_t nObj, FILE *fp);
size_t fwrite (void *ptr, size_t size, size_t nObj, FILE *fp);
```

Functions analyzed in C programming courses

Binary I/O often not introduced

The C Standard Library

- The I/O C standard is "fully buffered"
  - Each I/O is done only when the I/O buffer is full
  - Each "**flush**" operation writes the I/O buffer on the I/O device

```
#include <stdio.h>

void setbuf (FILE *fp, char *buf);

int fflush (FILE *fp);
```

The standard error is never buffered

For concurrent process, use
setbuf (stdout, 0) once
fflush (stdout) after each output instruction

The C Standard Library

- Instead of operating one character, one type, or one line at a time, we may want to operate on one ”structure” at a time
  - In this case we usually perform binary I/O

```
#include <stdio.h>

size_t fread (void *ptr, size_t size, size_t nObj, FILE *fp);

size_t fwrite (void *ptr, size_t size, size_t nObj, FILE *fp);
```

The C Standard Library

- Each (single) operation operates on an aggregate object of specific size
- Often used to manage binary files
  - Serialized (a single operation for the whole struct)
    - With getc/putc it would be necessary to iterate on all the fields of the struct
    - With gets/puts it is not possible, because both would terminate on NULL bytes or new-lines
  - Compatibility (inter-architecture) issues
    - Data format compatibility (e.g., integers, reals, etc.)
    - Different offsets for the fields of the struct

```
size_t fread (void *ptr, size_t size, size_t nObj, FILE *fp);
size_t fwrite (void *ptr, size_t size,size_t nObj, FILE *fp);
```

The C Standard Library

- C functions fread and fwrite are very similar to the UNIX system call read and write
  - We will analyze binary I/O directly in UNIX

```
size_t fread (void *ptr, size_t size, size_t nObj, FILE *fp);
size_t fwrite (void *ptr, size_t size,size_t nObj, FILE *fp);
```

Introduction

- Advanced I/O covers numerous topics and funtionalities
  - Encoding
    - ASCII, UNICODE, and Binary files
  - UNIX I/O
  - Filesystem manipulation
  - Non-blocking I/O
  - File locking
  - I/O multiplexing
  - Asynchronous I/O
  - Memory-mapped I/O

Part of "Operating Systems"
Only **reviewed** in SDP

Newly introduced in SDP
"Advanced I/O unit"