A C Primer (9): I/O and Files

Ion Mandoiu
Laurent Michel
Revised by M. Khan, J. Shi and W. Wei

errno

- Most C library functions can “fail”
  - When they do, they return a flag reporting failure… (-1)
  - Some set a global variable to report the exact error code

errno

```
// To use errno, include <errno.h>
```

- Check manual page to interpret the error code
- Print a more descriptive message with perror()

```
void perror(const char *str);
```

- Avoid functions that set errno in multithreaded code
  - Prefer thread-safe versions when available

Files and directories

- A file is an object that stores information, data, etc.

Example:

files you create with an editor (.c, .h, Makefile, readme, etc.)

executable generated by the compiler, and gcc itself

other devices, like screen, keyboard, …

- In Linux, files are organized in directories
  - A directory can have subdirectories and files
  - The top directory is /
- A path specifies the location of file/directory in the file system

/home/john

In Unix/Linux, everything is a file

The stdio library

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

- Declares FILE type and function prototypes
  - FILE is an **opaque type** (system dependent) for operating on files
    - It is a structure, but do not try to change it directly!
  - Use library functions to access FILE objects, via pointers (FILE *)
- Defines "standard" streams stdin, stdout, stderr $\leftarrow$ They are FILE *
  - Created automatically when program starts
  - They are files!
- The library is linked automatically by the compiler

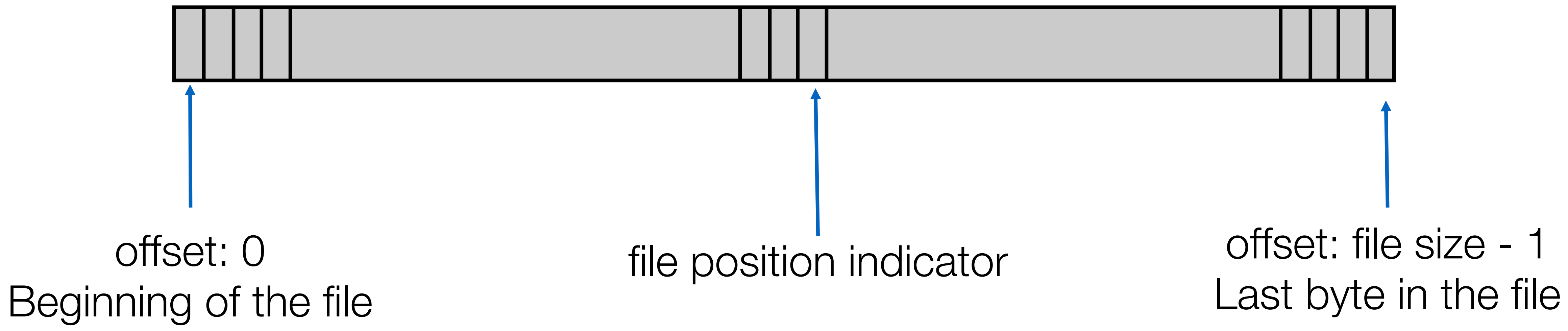
Files and I/O API

- In C, a file is simply a sequential stream of bytes
- The "f" family of functions (fopen, fclose, fread, fgetc, fscanf, fprintf,…) are C library functions to operate on files
    - All these use a FILE* abstraction to represent a file
    - The C library provides buffering
        - That's why sometimes you do not see output of printf immediately

We will learn another set of functions provided by OS

File as stream of bytes

- Before use a file must be "open"
  - This sets a **position indicator** for reading and/or writing
- Each read/write starts from current position, and moves the indicator
  - Writing after last byte increases the file size
- Position indicator can also be changed with fseek
- All open files are closed when program ends
  - Good practice to close explicitly when no longer needed



Opening Streams

FILE* fopen(const char *filename, const char *mode);

- Open the file filename in mode as a stream of bytes
- Returns a pointer to FILE (FILE *) or NULL (and errno is set)
- Mode
  - "r" : Reading mode
  - "r+" : Read and write
  - "w" : Writing mode, file is created or truncated to zero length
  - "w+" : Read and write, but the file is created or truncated
  - "a" : Append mode, the file is created if it does not exist
  - "a+" : Read and append, the file is created if it does not exist. Reading starts at beginning, but writing done at the end

Check return values!

Closing Streams

```
int fclose(FILE *stream);
```

- Close a stream
- Returns
    - 0 if it worked
    - EOF if there was a problem (and errno is set)

fgetc / fputc (one byte at a time)

```
int fgetc( FILE *stream);
int fputc(int c, FILE *stream);
```

- Read or write one (ASCII) character (8-bits) at a time
  - Can be slow for large files
- fgetc reads a character from the stream and returns the character just read in (as unsigned char extended to int)
    - Returns EOF when at the end of file or on error
- fputc writes the character received as argument to the stream and returns the character that was just written out
    - Returns EOF on error

getc / putc and ungetc

```
int getc(FILE *stream);
int putc(int c, FILE *stream);
```

- Same as fgetc/fputc except they may be implemented as macros
- Use fgetc/fputc unless you have strong reasons not to

```
int ungetc(int c, FILE *stream);
```

- Pushes last read char back to stream, where it is available for subsequent read operations
- Only one pushback guaranteed

getchar / putchar

```
int getchar(void)
// same as fgetc(stdin)
```

- Reads a character from stdin
- Returns the character just read in, or EOF on end-of-file or errors

```
int putchar(int c)
// same as fputc(c, stdout)
```

- Writes the character received as argument on stdout
- Returns the character that was just written out, or EOF on errors

More than one byte: get a line

```
char *fgets(char *buf, int size, FILE *in)
```

- Reads the next line from in into buffer buf
- Halts at ‘\n’ or after size-1 characters have been read
  - NUL is placed at the end
- Returns pointer to buf if ok, NULL otherwise
- Do not use gets(char *)! – buffer overflow

```
int fputs(const char *str, FILE *out)
```

- Writes the string str to out, stopping at ‘\0’
- Returns number of characters written or EOF

Formatted output

```
int fscanf(FILE *stream, const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
```

- Formatted input from file and output to file
- Like scanf()/printf(), but not from stdin or to stdout

For binary data

```
size_t fread (void *ptr, size_t sz, size_t n, FILE *stream);
size_t fwrite(void *ptr, size_t sz, size_t n, FILE *stream);
```

- Read / write a sequence of byte from/to a stream
- Return the number of items read or written
  - If smaller than n, EOF or error

Example:

```
int    A[10][20];
size_t    n = 10 * 20;
if (fwrite(A, sizeof(int), n, fp) != n) {
   // error
```

Moving file position indicator

Not all streams are seekable
errno: EBADF

```
long ftell(FILE *stream);
```

- Read file position indicator
- Return -1 on error

```
int fseek(FILE * stream, off_t offset, int whence);
```

- Set the file position indicator
- Return 0 on success and -1 on error

Example:

```
fseek(fp,   0, SEEK_SET); // move to the beginning
fseek(fp, 200, SEEK_CUR); // move forward 200 bytes
fseek(fp,  -1, SEEK_END); // move to the last byte
```

More useful stdio functions

```
//Check if end-of-file is set (after a read attempt!)
int feof(FILE * stream);

//Force write of buffered data
int fflush(FILE * stream);
```

Read the manual pages!

Check the return values!