C File Processing

Streams, Files, Reading from and Writing to Files, Buffered Access

SoftUni Team

Technical Trainers

Software University

http://softuni.bg

Table of Contents

SOFTWARE UNIVERSITY FOUNDATION

Files

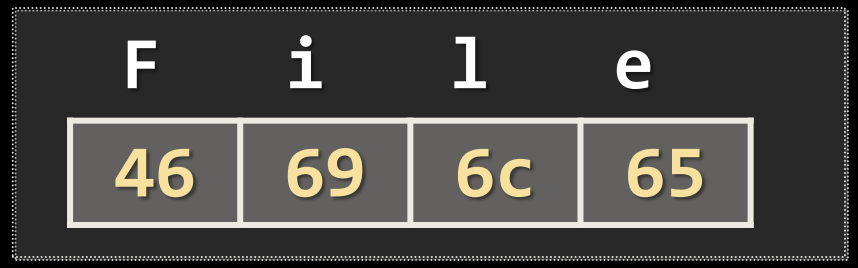
What are Files?

Files

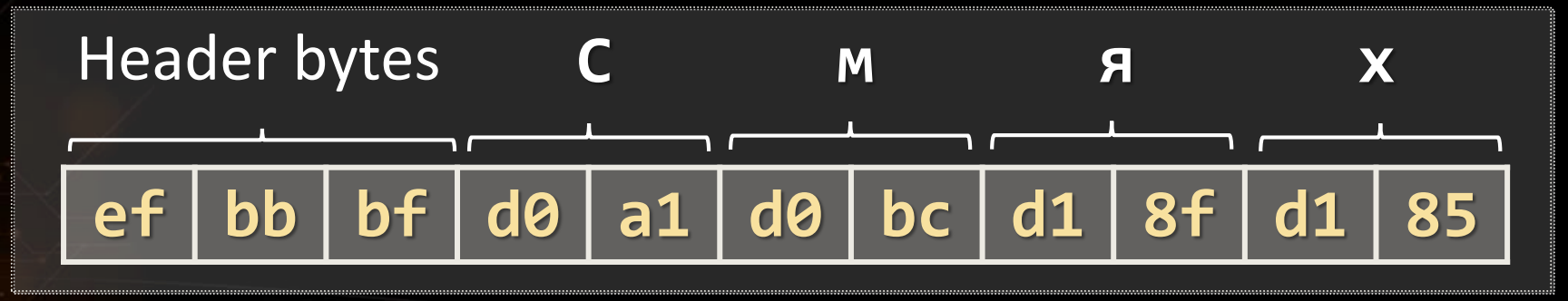
- A file is a resource for storing information
  - Located on a storage device (e.g. hard-drive)
  - Has name, size, extension and contents
  - Stores information as series of bytes
- Two file types – text and binary

Text Files

- Text files contain text information
  - Store text differently according to the encoding
    - E.g. in ASCII (0..127 codes) a character is represented by 1 byte

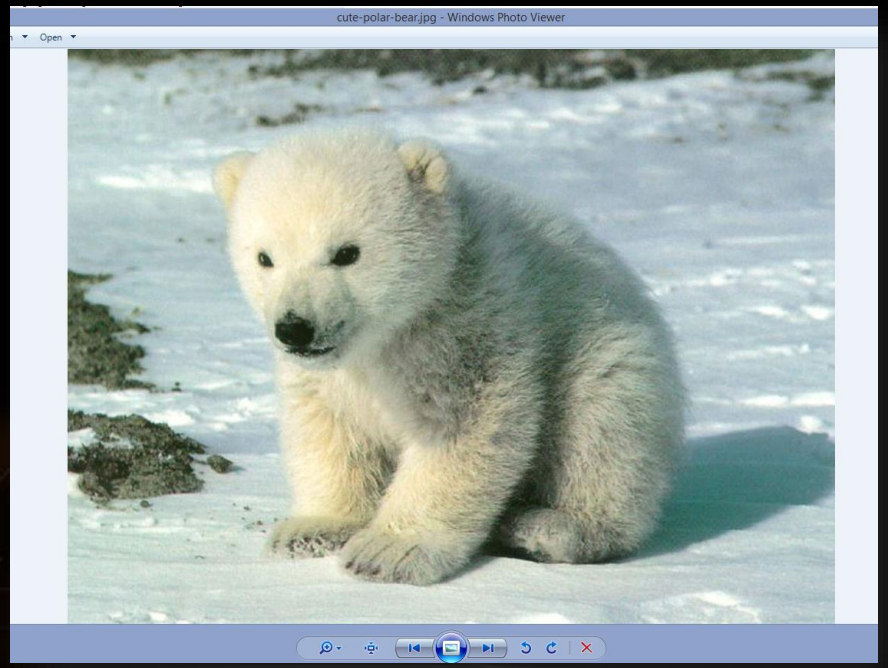


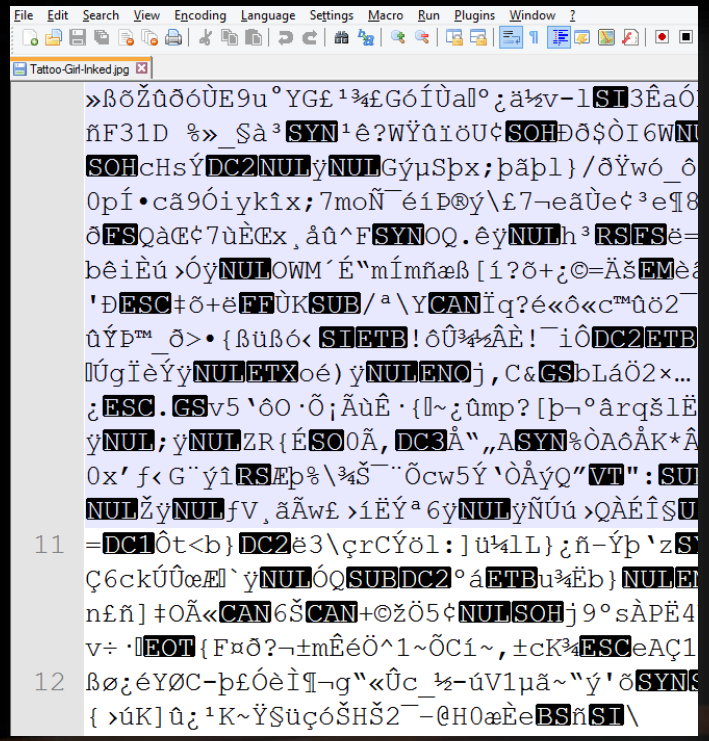
    - In UTF8 (0..65535 codes) a character is represented by 1-4 bytes



Binary Files

- Binary files store raw sequences of bytes
  - Can contain any data (images, sound, multimedia, etc.)
  - Not human-readable





SOFTWARE UNIVERSITY FOUNDATION

Text and Binary Files

Live Demo in Hex Editor

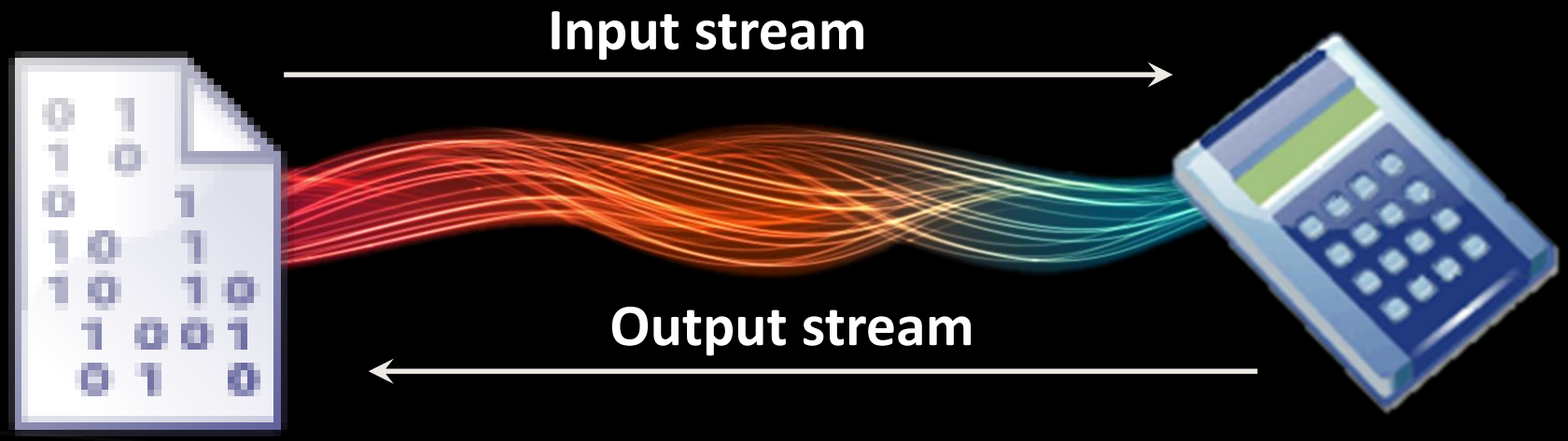
SOFTWARE UNIVERSITY FOUNDATION

What Is Stream?

Streams Basic Concepts

What is Stream?

- Stream is the natural way to transfer data in the computer world
- To read or write a file, we open a stream connected to the file and access the data through the stream



Streams Basics

- Streams are means for transferring (reading and writing) data into and from devices
- Streams are ordered sequences of bytes
  - Provide consecutive access to its elements
- Different types of streams are available to access different data sources:
  - File access, network access, memory streams and others
- Streams are opened before using them and closed after that

Stream – Example

- Position is the current position in the stream
- The buffer keeps the current **position + n** bytes of the stream

Files in C

- C view each file as a sequential stream of bytes
- Opening files returns a pointer to a **`FILE`** structure

```
FILE *file = fopen("file.txt", "r");
```

  - Keeps data about the file and the state of the stream
  - Processed with functions from **`<stdio.h>`** header
  - In Linux, an opened file has a file descriptor associated with it
    - File descriptors are kernel-level integers mapped to specific files
    - Streams provide an abstraction over working with file descriptors

Opening Files

- `FILE *fopen(const char *path, const char *mode)` – opens a file specified by its `path` in a given `mode`
  - Returns a pointer to a `FILE` structure, holding data about the file
  - Return `NULL` if the file does not exist / cannot be opened

```
FILE *file = fopen("file.txt", "r");
if (file != NULL)
{
    // Process file...
    fclose(file);
}
```

Open the file in read mode

Closes the file stream after the program is done using it

File Modes

- Files can be opened in the following modes:

Mode	Action	Description
r	read	Starts at beginning (file must exist)
w	write	Overwrites or creates new file
a	append	Starts at end (file may not exist)
r+	read + write	Starts at beginning (file must exist)
w+	read + write	Overwrite or create new file
a+	read + write	Starts at end (file may not exist)

Closing Files

- `int fclose(FILE *stream)` – flushes the stream (writes the internal buffer) and closes the underlying file descriptor

```
FILE *file = fopen("program.c", "w");
if (file != NULL)
{
    ...
    fclose(file);
}
```

  - Files should only be closed if they were successfully opened

Reading File Characters

- `int fgetc(FILE *stream)` – reads and returns the next character in the stream as `unsigned char` (cast to `int`)
  - Returns `EOF` (end of file) when end is reached

```
FILE *file = fopen("file.txt", "r");
if (file)
{
    int ch1 = fgetc(); // T
    int ch2 = fgetc(); // e
    ...
    fclose(file);
}
```

file.txt

```
Texting is
the act of
sending a
message
via ...
```

Reading a Text File

```
FILE *file = fopen("file.txt", "r");
if (file != NULL)
{
    char c;
    while (1)
    {
        c = fgetc(file);
        if (c == EOF)
            break;
        printf("%c", c);
    }

    fclose(file);
}
```

Warning: This is very slow!

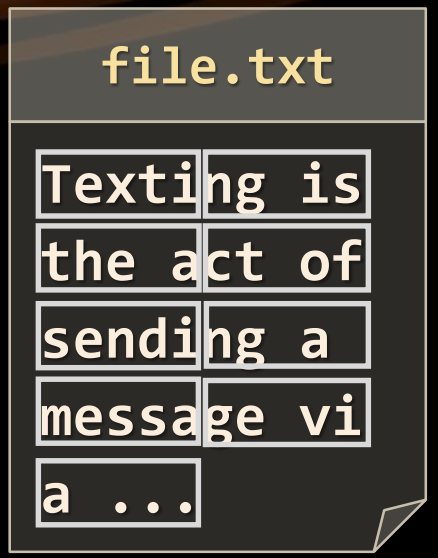
Reading Data With Buffer

- Reading character by character is slow
  - The OS has to call the storage device driver for every single byte
  - …and reading/writing to HDD is very slow
- `size_t fread(void *buffer, size_t size, size_t n, FILE *stream)` – reads `n` elements of data, each `size` bytes long from `*stream` into `*buffer`
  - Returns the number of bytes read
  - Returns `0` if there is no more data to be read

Using fread() – Example

```
#define BUFFER_SIZE 5

int main()
{
    FILE *file = fopen("file.txt", "r");
    if (file)
    {
        char buffer[BUFFER_SIZE + 1];
        while (!feof(file)
        {
            int readBytes =
                fread(buffer, 1, BUFFER_SIZE, file);
            buffer[readBytes] = '\0';
            printf("%s\n", buffer);
        }
    }
```



Reads bytes in portions, each **BUFFER_SIZE** long

End of File and Error

- `int feof(FILE *stream)` – checks if end of file is reached

```
while (!feof(srcFile))
{
    size_t readBytes = fread(buffer, 1, BUFFER_SIZE, srcFile);

    printf("%s\n", buffer);
}
```

- `int ferror(FILE *stream)` – checks whether the error indicator has been set for `*stream` (i.e. there is an error)

Writing to Files

- Just like reading, writing to files should be buffered
  - Minimizes hard-drive overhead
  - The optimal buffer size is a multiple of **4096**
    - The file system usually keeps data in clusters of 4096 bytes (4KB)
    - CPU can easily cache read data
- **size_t fwrite(const void *buffer, size_t size, size_t n, FILE *stream)** – write **n** elements of data, each **size** bytes long from ***buffer** into ***stream**

File Copy Console Program

SOFTWARE UNIVERSITY FOUNDATION

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

#define BUFFER_SIZE 4096

void copy(const char *srcPath, const char *destPath);
void die(const char *msg);

int main(int argc, char **argv)
{
    if (argc < 3)
        die("Usage: ./prog <src-file> <dest-file>");

    copy(argv[1], argv[2]);
    printf("File copied\n");

    return (EXIT_SUCCESS);
}
```

File Copy Console Program (2)

```
void die(const char *msg)
{
    if (errno)
    {
        perror(msg);
    }
    else
    {
        printf("ERROR: %s\n", msg);
    }

    exit(1);
}
```

Terminates program

File Copy Console Program (3)

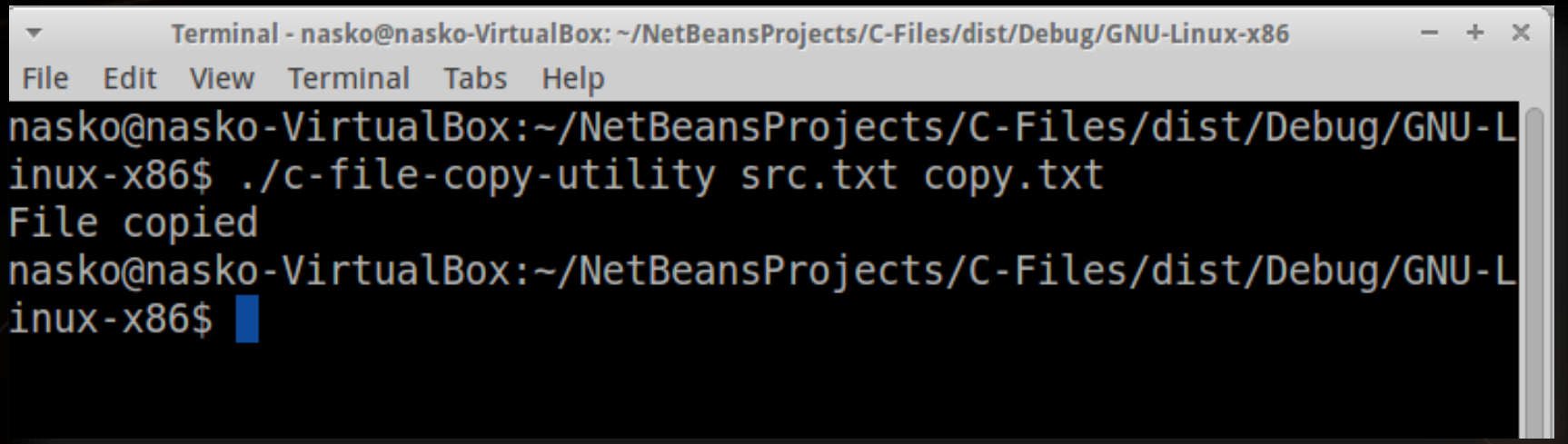
SOFTWARE UNIVERSITY FOUNDATION

```
void copy(const char *srcPath, const char *destPath)
{
    FILE *srcFile = fopen(srcPath, "r");
    if (!srcFile) die(NULL);

    FILE *destFile = fopen(destPath, "w");
    if (!destFile) die(NULL);

    char buffer[BUFFER_SIZE];
    while (!feof(srcFile))
    {
        size_t readBytes = fread(buffer, 1, BUFFER_SIZE, srcFile);
        fwrite(buffer, 1, readBytes, destFile);
    }

    fclose(destFile);
    fclose(srcFile);
}
```



File Copy Utility

Live Demo

Reading Lines From File

- `ssize_t getline(char **line, size_t *n, FILE *stream)`
  – reads next line to end and writes it to `**line`
  - Line end is denoted by `'\n'`
- Two use cases:
  1. Pass own heap-allocated `line` and length `*n`
     - `getline()` will write to `line` and `realloc()` if buffer is too small
  2. Pass `NULL` pointer as line
     - `getline()` allocates its own buffer and assigns it to `line`

Reading Lines – Example

```
FILE *file = fopen("file.txt", "r");
if (file)
{
    char *line;
    size_t length = 0;

    while (1)
    {
        ssize_t readBytes = getline(&line, &length, file);
        if (feof(file) && readBytes <= 0)
            break;

        printf("Read bytes: %d\n", readBytes);
        printf("Line size: %d\n", length);
        printf("%s\n", line);
    }

    free(line);
}
```

Free the memory
`malloc()`'d by `getline()`

Writing Formatted Text to File

- `int fprintf(FILE *stream, const char *format, …)` – formats the strings and writes the result to `*stream`

```
FILE *file = fopen("main.c", "a");
if (file)
{
    fprintf(file, "\n // End of file");
    fclose(file);
}
```

Fixing Movie Subtitles

Live Demo

C Programming – File Processing

SOFTWARE UNIVERSITY FOUNDATION

Questions?

https://softuni.bg/trainings/1212/C-Programming-October-2015

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license

- Attribution: this work may contain portions from
  - "Programming Basics" course by Software University under CC-BY-SA license

Free Trainings @ Software University

- Software University Foundation – softuni.org
- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University @ YouTube
  - youtube.com/SoftwareUniversity
- Software University Forums – forum.softuni.bg