

# ECSE 427/COMP310 Lab6

Jason Zixu Zhou

Oct-4-2024

# Basic Concepts of File Operations

- **File Types:**
  - **Text Files:** Store data in readable text format.
  - **Binary Files:** Store data in binary format, not human-readable.
- **Concept of File Streams:**
  - **Definition:** A stream represents a sequence of bytes.
  - **Visual Aid:** Diagram showing a stream flowing from a file to a program.

# Opening Files

- **Using fopen() Function:**
  - **Syntax:** FILE \*fopen(const char \*filename, const char \*mode);
  - **Parameters:** Explain filename and mode.
- **File Modes Explained:**
  - **"r":** Open for reading. If the file does not exist, the opening fails.
  - **"w":** Open for writing. If the file exists, it is truncated to zero length.
  - **"a":** Open for appending. The file is created if it does not exist.
  - **"r+":** Open for reading and writing from the beginning.

```
#include <stdio.h>

int main()
{
    // Attempt to open a file in read-only mode
    FILE *file = fopen("example.txt", "r");
    // Check if the file was successfully opened
    if (file == NULL)
    {
        perror("Error opening file"); // Print error message if opening fails
        return -1;
    }
    // Additional file operations go here
    fclose(file); // Close the file
    return 0;
}
```

# Reading Files

- **Reading Functions:**

- **fgetc():** Reads a single character.
- **fgets():** Reads a string until newline or EOF.
- **fread():** Reads multiple bytes, used in binary files.

Lab6 C Files > C read.c > main()

```
1  #include <stdio.h>
2
3  int main()
4  {
5      // Open file for reading
6      FILE *file = fopen("example.txt", "r");
7      // Check if the file is opened successfully
8      if (file == NULL)
9      {
10         perror("Error opening file");
11         return -1;
12     }
13     char line[100];
14     // Read the file line by line until end of file
15     while (fgets(line, sizeof(line), file) != NULL)
16     {
17         printf("%s", line); // Print each line
18     }
19     fclose(file); // Close the file
20     return 0;
21 }
```



# Fread Source Code Analysis

- <https://sourceware.org/git/?p=glibc.git;a=blob;f=libio/iofread.c;h=3294fac5f5911d78f8f9d607e5b973a08c6727b2;hb=refs/heads/release/2.40/master>

```
27 #include "libioP.h"
28
29 size_t
30 _IO_fread (void *buf, size_t size, size_t count, FILE *fp)
31 {
32     size_t bytes_requested = size * count;
33     size_t bytes_read;
34     CHECK_FILE (fp, 0);
35     if (bytes_requested == 0)
36         return 0;
37     _IO_acquire_lock (fp);
38     bytes_read = _IO_sgetn (fp, (char *) buf, bytes_requested);
39     _IO_release_lock (fp);
40     return bytes_requested == bytes_read ? count : bytes_read / size;
41 }
42 libc_hidden_def (_IO_fread)
43
44 weak_alias (_IO_fread, fread)
45
46 # ifndef _IO_MTSAFE_IO
47 strong_alias (_IO_fread, __fread_unlocked)
48 libc_hidden_def (__fread_unlocked)
49 weak_alias (_IO_fread, fread_unlocked)
50 # endif
```

# Line-by-Line Code Analysis:

```
29 size_t
30 _IO_fread (void *buf, size_t size, size_t count, FILE *fp)
31 {
```

`_IO_fread` function: the internal implementation of `fread`.

`buf` is the destination buffer for the data read

`size` is the size of each data element

`count` is the number of elements to read

`fp` is a pointer to a `FILE` structure representing the file stream

# Line-by-Line Code Analysis:

```
27 #include "libioP.h"
28
29 size_t
30 _IO_fread (void *buf, size_t size, size_t count, FILE *fp)
31 {
32     size_t bytes_requested = size * count;
33     size_t bytes_read;
34     CHECK_FILE (fp, 0);
35     if (bytes_requested == 0)
36         return 0;
37     _IO_acquire_lock (fp);
38     bytes_read = _IO_sgetn (fp, (char *) buf, bytes_requested);
39     _IO_release_lock (fp);
40     return bytes_requested == bytes_read ? count : bytes_read / size;
41 }
42 libc_hidden_def (_IO_fread)
43
44 weak_alias (_IO_fread, fread)
45
46 # ifndef _IO_MTSAFE_IO
47 strong_alias (_IO_fread, __fread_unlocked)
48 libc_hidden_def (__fread_unlocked)
49 weak_alias (_IO_fread, fread_unlocked)
50 # endif
```



# Line-by-Line Code Analysis:

```
27 #include "libioP.h"
28
29 size_t
30 _IO_fread (void *buf, size_t size, size_t count, FILE *fp)
31 {
32     size_t bytes requested = size * count;
33     size_t bytes read;
34     CHECK_FILE (fp, 0);
35     if (bytes_requested == 0)
36         return 0;
37     _IO_acquire_lock (fp);
38     bytes_read = _IO_sgetn (fp, (char *) buf, bytes_requested);
39     _IO_release_lock (fp);
40     return bytes_requested == bytes_read ? count : bytes_read / size;
41 }
42 libc_hidden_def (_IO_fread)
43
44 weak_alias (_IO_fread, fread)
45
46 # ifndef _IO_MTSafe_IO
47 strong_alias (_IO_fread, __fread_unlocked)
48 libc_hidden_def (__fread_unlocked)
49 weak_alias (_IO_fread, fread_unlocked)
50 # endif
```



# Line-by-Line Code Analysis:

```
27 #include "libioP.h"
28
29 size_t
30 _IO_fread (void *buf, size_t size, size_t count, FILE *fp)
31 {
32     size_t bytes_requested = size * count;
33     size_t bytes_read;
34     CHECK_FILE (fp, 0);
35     if (bytes_requested == 0)
36         return 0;
37     _IO_acquire_lock (fp);
38     bytes_read = _IO_sgetn (fp, (char *) buf, bytes_requested);
39     _IO_release_lock (fp);
40     return bytes_requested == bytes_read ? count : bytes_read / size;
41 }
42 libc_hidden_def (_IO_fread)
43
44 weak_alias (_IO_fread, fread)
45
46 # ifndef _IO_MTSAFE_IO
47 strong_alias (_IO_fread, __fread_unlocked)
48 libc_hidden_def (__fread_unlocked)
49 weak_alias (_IO_fread, fread_unlocked)
50 # endif
```

# Line-by-Line Code Analysis:

```
27 #include "libioP.h"
28
29 size_t
30 _IO_fread (void *buf, size_t size, size_t count, FILE *fp)
31 {
32     size_t bytes_requested = size * count;
33     size_t bytes_read;
34     CHECK_FILE (fp, 0);
35     if (bytes_requested == 0)
36         return 0;
37     _IO_acquire_lock (fp);
38     bytes_read = _IO_sgetn (fp, (char *) buf, bytes_requested);
39     _IO_release_lock (fp);
40     return bytes_requested == bytes_read ? count : bytes_read / size;
41 }
42 libc_hidden_def (_IO_fread)
43
44 weak_alias (_IO_fread, fread)
45
46 # ifdef _IO_MTSafe_IO
47 strong_alias (_IO_fread, __fread_unlocked)
48 libc_hidden_def (__fread_unlocked)
49 weak_alias (_IO_fread, fread_unlocked)
50 # endif
```

# Line-by-Line Code Analysis:

```
27 #include "libioP.h"
28
29 size_t
30 _IO_fread (void *buf, size_t size, size_t count, FILE *fp)
31 {
32     size_t bytes_requested = size * count;
33     size_t bytes_read;
34     CHECK_FILE (fp, 0);
35     if (bytes_requested == 0)
36         return 0;
37     _IO_acquire_lock (fp);
38     bytes_read = _IO_sgetn (fp, (char *) buf, bytes_requested);
39     _IO_release_lock (fp);
40     return bytes_requested == bytes_read ? count : bytes_read / size;
41 }
42 libc_hidden_def (_IO_fread)
43
44 weak_alias (_IO_fread, fread)
45
46 # ifdef _IO_MTSafe_IO
47 strong_alias (_IO_fread, __fread_unlocked)
48 libc_hidden_def (__fread_unlocked)
49 weak_alias (_IO_fread, fread_unlocked)
50 # endif
```



# Line-by-Line Code Analysis:

```
27 #include "libioP.h"
28
29 size_t
30 _IO_fread (void *buf, size_t size, size_t count, FILE *fp)
31 {
32     size_t bytes_requested = size * count;
33     size_t bytes_read;
34     CHECK_FILE (fp, 0);
35     if (bytes_requested == 0)
36         return 0;
37     _IO_acquire_lock (fp);
38     bytes_read = _IO_sgetn (fp, (char *) buf, bytes_requested);
39     _IO_release_lock (fp);
40     return bytes_requested == bytes_read ? count : bytes_read / size;
41 }
42 libc_hidden_def (_IO_fread)
43
44 weak_alias (_IO_fread, fread)
45
46 # ifndef _IO_MTSafe_IO
47 strong_alias (_IO_fread, __fread_unlocked)
48 libc_hidden_def (__fread_unlocked)
49 weak_alias (_IO_fread, fread_unlocked)
50 # endif
```

# Line-by-Line Code Analysis:

```
27 #include "libioP.h"
28
29 size_t
30 _IO_fread (void *buf, size_t size, size_t count, FILE *fp)
31 {
32     size_t bytes_requested = size * count;
33     size_t bytes_read;
34     CHECK_FILE (fp, 0);
35     if (bytes_requested == 0)
36         return 0;
37     _IO_acquire_lock (fp);
38     bytes_read = _IO_sgetn (fp, (char *) buf, bytes_requested);
39     _IO_release_lock (fp);
40     return bytes_requested == bytes_read ? count : bytes_read / size;
41 }
42 libc_hidden_def (_IO_fread)
43
44 weak_alias (_IO_fread, fread)
45
46 # ifndef _IO_MTSAFE_IO
47 strong_alias (_IO_fread, __fread_unlocked)
48 libc_hidden_def (__fread_unlocked)
49 weak_alias (_IO_fread, fread_unlocked)
50 # endif
```

# Writing Files

- **Writing Functions:**

- **fputc():** Writes a single character.
- **fputs():** Writes a string.
- **fwrite():** Writes multiple bytes, used in binary files.

Lab6 C Files > C write.c > ...

```
1  #include <stdio.h>
2
3  int main()
4  {
5      // Open file for writing
6      FILE *file = fopen("output.txt", "w");
7      // Check if the file opens successfully
8      if (file == NULL)
9      {
10         perror("Error opening file");
11         return -1;
12     }
13     const char *text = "Hello, World!\n";
14     fputs(text, file); // Write text to the file
15     fclose(file);      // Close the file
16     return 0;
17 }
```





# Closing Files

- **Using fclose() Function:**
  - **Syntax:** `int fclose(FILE *stream);`
  - **Purpose:** Ensure all data is written and resources are released.
- **Importance:**
  - **Data Integrity:** Prevents data loss.
  - **Resource Management:** Frees up system resources.

# Example: Simple File Read/Write

Lab6 C Files > C example\_rw.c > ...

```
1  #include <stdio.h>
2
3  void copyFileContent(FILE *source, FILE *target)
4  {
5      char buffer[256];
6      // Read from the source file and write to the target file
7      while (fgets(buffer, sizeof(buffer), source) != NULL)
8      {
9          fputs(buffer, target);
10     }
11 }
12
13 int main()
14 {
15     // Open the input file
16     FILE *inputFile = fopen("input.txt", "r");
17     // Check if the input file opens successfully
18     if (inputFile == NULL)
19     {
20         perror("Error opening input file");
21         return -1;
22     }
23     // Open the output file
24     FILE *outputFile = fopen("output.txt", "w");
25     // Check if the output file opens successfully
26     if (outputFile == NULL)
27     {
28         fclose(inputFile); // Close the input file
29         perror("Error opening output file");
30         return -1;
31     }
32
33     // Copy the content from input to output
34     copyFileContent(inputFile, outputFile);
35
36     // Close both files
37     fclose(inputFile);
38     fclose(outputFile);
```

# Header Files

- **Purpose of Header Files:**
  - **Declare Functions:** Ensure function calls are recognized.
  - **Define Macros:** Simplify repetitive code.
  - **Define Data Structures:** Share structures across files.
- **Common Headers:**
  - **List:** Includes `<stdio.h>`, `<stdlib.h>`, and explanation of their common uses.



# Creating Custom Header Files

- **How to Create and Use:**

- **Creating .h Files:** Tips on writing your own header files.
- **Include Guards:** `#ifndef`, `#define`, `#endif` to prevent duplicate inclusion.

- **Example:**

```
Lab6 C Files > C myfunctions.h > ...
1  #ifndef MYFUNCTIONS_H
2  #define MYFUNCTIONS_H
3
4  #include <stdio.h>
5
6  void greet(void)
7  {
8      printf("Hello, welcome to the program!\n"); // Define a greeting function
9  }
10
11 #endif // MYFUNCTIONS_H
```

```
Lab6 C Files > C example_use_head.c > ...
1  #include "myfunctions.h"
2
3  int main()
4  {
5      greet(); // Call the greeting function defined in the header file
6      return 0;
7  }
8
```

# #pragma once

- **What is #pragma once?**
- A preprocessor directive used to prevent multiple inclusions of the same header file.
- **Benefits:**
  - Simpler and less error-prone than traditional include guards.
  - Supported by most modern compilers (e.g., GCC, Clang, MSVC).

Lab6 C Files > C myheader.h > ...

```
1 // myheader.h
2 #pragma once
3
4 // Function declarations
5 void myFunction();
6 void anotherFunction();
7
```

C open.c

C myheader.h U

C myheader.c U X

Lab6 C Files > C myheader.c > anotherFunction()

```
1 // main.c
2 #include <stdio.h>
3 #include "myheader.h"
4 #include "myheader.h" // Even if this is included twice
5
6 void myFunction()
7 {
8     printf("Hello from myFunction!\n");
9 }
10
```

# Opening Binary Files

- **Using `fopen()` for Binary Files:**
- Append "b" to the mode string in `fopen()` function, e.g., "rb", "wb", "ab", "rb+", "wb+", "ab+".