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.

```c
#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()

```c
#include <stdio.h>

int main()
{
    // Open file for reading
    FILE *file = fopen("example.txt", "r");
    // Check if the file is opened successfully
    if (file == NULL)
    {
        perror("Error opening file");
        return -1;
    }
    char line[100];
    // Read the file line by line until end of file
    while (fgets(line, sizeof(line), file) != NULL)
    {
        printf("%s", line); // Print each line
    }
    fclose(file); // Close the file
    return 0;
}
```

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 # 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 # 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 > ...

```c
#include <stdio.h>

int main()
{
    // Open file for writing
    FILE *file = fopen("output.txt", "w");
    // Check if the file opens successfully
    if (file == NULL)
    {
        perror("Error opening file");
        return -1;
    }
    const char *text = "Hello, World!\n";
    fputs(text, file); // Write text to the file
    fclose(file);      // Close the file
    return 0;
}
```

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 > ...

```c
#include <stdio.h>

void copyFileContent(FILE *source, FILE *target)
{
    char buffer[256];
    // Read from the source file and write to the target file
    while (fgets(buffer, sizeof(buffer), source) != NULL)
    {
        fputs(buffer, target);
    }
}

int main()
{
    // Open the input file
    FILE *inputFile = fopen("input.txt", "r");
    // Check if the input file opens successfully
    if (inputFile == NULL)
    {
        perror("Error opening input file");
        return -1;
    }
    // Open the output file
    FILE *outputFile = fopen("output.txt", "w");
    // Check if the output file opens successfully
    if (outputFile == NULL)
    {
        fclose(inputFile); // Close the input file
        perror("Error opening output file");
        return -1;
    }

    // Copy the content from input to output
    copyFileContent(inputFile, outputFile);

    // Close both files
    fclose(inputFile);
    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 > ...

```c
#ifndef MYFUNCTIONS_H
#define MYFUNCTIONS_H

#include <stdio.h>

void greet(void)
{
    printf("Hello, welcome to the program!\n"); // Define a greeting function
}

#endif // MYFUNCTIONS_H
```

Lab6 C Files > C example_use_head.c > ...

```c
#include "myfunctions.h"

int main()
{
    greet(); // Call the greeting function defined in the header file
    return 0;
}

```

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+".