Assignment 3 – XD Report Template

Debi Majumdar

CSE 13S – Winter 24

Purpose

Audience for this section: Pretend that you are working in industry, and write this paragraph for your boss. You are answering the basic question, "What does this thing do?". This section can be short. A single paragraph is okay.

Do not just copy the assignment PDF to complete this section, use your own words.

This program reads files and displays their contents in a format similar to xxd. It implements a basic buffered reader to read files in 16-byte chunks without using buffered I/O functions like 'fread'. It opens a specified file, reads its contents, and prints them in a structured format including hexadecimal representation and ASCII characters. It is useful for examining and interpreting binary files.

Questions

Please answer the following questions before you start coding. They will help guide you through the assignment. To make the grader's life easier, please do not remove the questions, and simply put your answers below the text of each question.

- What is a buffer? Why use one?

  A buffer is a temporary storage area for data. It's used to hold information while it's being transferred between locations, such as from a file to memory or between processes. Buffers help improve performance by reducing the number of I/O operations, as data can be read or written in chunks rather than individually.

- What is the return value of `read()`? What are the inputs?

  The `read()` function returns the number of bytes actually read. It returns 0 if the end-of-file (EOF) is reached, -1 on error, and otherwise returns the number of bytes read. The inputs to `read()` are the file descriptor (`int fd`), a buffer to store the data (`void *buf`), and the number of bytes to read (`size_t count`).

- What is a file no. ? What are the file numbers of `stdin`, `stdout`, and `stderr`?

  A file descriptor, or "file no.", is an integer value that represents an open file within a process. In Unix-like systems, stdin has file number 0, stdout has file number 1, and stderr has file number 2. These represent the standard input, standard output, and standard error streams respectively.

- What are the cases in which `read(0,16)` will return 16? When will it *not* return 16?

  `read(0,16)` will return 16 if there are at least 16 bytes available to be read from the file with file descriptor 0 (typically `stdin`). It will not return 16 if there are fewer than 16 bytes available, or if an error occurs during the read operation.

- Give at least 2 (very differnt) cases in which a file can not be read all at once

  1. If the file size exceeds the available memory (RAM) on the system, attempting to read the entire file into memory at once may result in an out-of-memory error. 2. In a networked environment, if the

file is located on a remote server and the network connection is unreliable or slow, attempting to read the entire file at once may encounter network timeouts or interruptions. In such cases, reading the file in smaller chunks or streaming it may be more appropriate.

Testing

List what you will do to test your code. Make sure this is comprehensive. [1] Be sure to test inputs with delays.

To thoroughly test the `xd.c` program, various comprehensive steps will be undertaken. Basic functionality will be assessed with different input files, including edge cases like empty files and minimal content. Error handling will be examined with invalid file names and unsupported types. Buffered reading will be tested with large files and partial buffers. Interactive and delay testing will ensure proper behavior with manual and delayed input. Performance will be measured with large files, and boundary and concurrency testing will evaluate extreme conditions like delays. Compatibility across platforms will ensure consistent behavior across all read world scenarios.

How to Use the Program

Audience: Write this section for the user of your program. You are answering the basic question, "How do I use this thing?". Don't copy the assignment exactly; explain this in your own words. This section will be longer for a more complicated program and shorter for a less complicated program. You should show how to compile and run your program. You should also describe any optional flags or inputs that your program uses, and what they do.

To show "code font" text within a paragraph, you can use `\lstinline{}`, which will look like this: `text`.

For a code block, use `\begin{lstlisting}` and `\end{lstlisting}`, which will look like this:

```
Here is some code in lstlisting.
```

And if you want a box around the code text, then use `\begin{lstlisting}[frame=single]` and `\end{lstlisting}`

which will look like this:

```
Here is some framed code (lstlisting) text.
```

Want to make a footnote? Here's how.[2]

Do you need to cite a reference? You do that by putting the reference in the file `bibtex.bib`, and then you cite your reference like this[1][2][3].

Using the `xd.c` program is simple. First, compile the source code with a C compiler like GCC or Clang using the command `gcc -o xd xd.c`. Then, run the program by providing the filename of the file you want to examine as a command-line argument, like `./xd filename`. If no filename is provided, the program reads from stdin. Optionally, you can include flags to customize the output. Overall, `xd.c` offers an easy-to-use solution for displaying file contents in a human-readable format, making it convenient for users to examine and interpret binary files.

Program Design

Audience: Write this section for someone who will maintain your program. In industry you maintain your own programs, and so your audience could be future you! List the main data structures and the main algorithms. You are answering the basic question, "How is this thing organized so that I can have a chance of fixing it?". This section will be longer for a more complicated program and shorter for a less complicated program.

[1] This question is a whole lot more vague than it has been the last few assignments. Continue to answer it with the same level of detail and thought.

[2] This is my footnote.

For maintenance purposes, the `xd.c` program mainly relies on a buffer as its primary data structure, used to temporarily store data read from files in 16-byte chunks. It utilizes file descriptors, employing functions like `open()` and `read()` for file I/O operations. The main algorithm revolves around a loop that reads and processes data from files incrementally. Within this loop, sub-algorithms handle the formatting and printing of hexadecimal and ASCII representations. Understanding this organization is crucial for efficient troubleshooting and debugging.

Pseudocode

Give the reader a top down description of your code! How will you break it down? What features will your code have? How will you implement each function.

My pseudocode for xd.c: 1. Function display_hex_ascii(data, length): a. For each byte in data: i. Print the hexadecimal representation of the byte. ii. If the byte index is odd, print a space. b. Pad with spaces if necessary to align with BUFFER_SIZE. c. Print a space. d. For each byte in data: i. If the byte is a printable ASCII character, print it. ii. Otherwise, print a period.

2. Function main(argc, argv): a. Initialize file_descriptor. b. If argc is 2: i. Open the file specified by argv[1]. ii. If the file cannot be opened, exit with error. c. Else if argc is 1: i. Set file_descriptor to STDIN_FILENO. d. Else: i. Exit with error. e. Initialize buffer of size BUFFER_SIZE. f. Initialize bytesRead. g. Initialize curr_byte to 0. h. While bytesRead = read(file_descriptor, buffer, BUFFER_SIZE) ¿ 0: i. If file_descriptor is STDIN_FILENO: 1. While bytesRead ¡ BUFFER_SIZE: a. Read remaining bytes from stdin into buffer. 2. Print the formatted output using display_hex_ascii. ii. Else: 1. Print the formatted output using display_hex_ascii. iii. Update curr_byte. iv. If bytesRead ¡ BUFFER_SIZE, break the loop. i. If file_descriptor is not STDIN_FILENO, close the file. j. Exit with success.

Function Descriptions

For each function in your program, you will need to explain your thought process. This means doing the following

- The inputs of every function (even if it's not a parameter)
- The outputs of every function (even if it's not the return value)
- The purpose of each function, a brief description about a sentence long.
- For more complicated functions, include pseudocode that describes how the function works
- For more complicated functions, also include a description of your decision making process; why you chose to use any data structures or control flows that you did.

Do not simply use your code to describe this. This section should be readable to a person with little to no code knowledge. **DO NOT JUST PUT THE FUNCTION SIGNATURES HERE. MORE EXPLANATION IS REQUIRED.**

Function display_hex_ascii(data, length):

This function is responsible for displaying the hexadecimal and ASCII representations of a given data buffer. It iterates through each byte in the data buffer. For each byte, it prints the hexadecimal representation followed by a space if the byte index is odd. After printing the hexadecimal values, it pads with spaces to align with the buffer size. It then prints a space and iterates through each byte again. For each byte, it prints the ASCII representation if it is a printable ASCII character, otherwise it prints a period.

Function main(argc, argv):

This function represents the main logic of the program. It takes command-line arguments argc (argument count) and argv (argument vector). It initializes the file descriptor and opens the specified file if provided as an argument, or sets the file descriptor to STDIN_FILENO if no filename is provided. It initializes a buffer to store data read from the file. It enters a loop to read data from the file descriptor in chunks of BUFFER_SIZE bytes. If the file descriptor is STDIN_FILENO, it ensures that the buffer is filled completely before processing. It calls the display_hex_ascii function to print the formatted output. After reading and processing the data, it closes the file if it was opened. Finally, it exits with success.

Optimizations

This section is optional, but is required if you do the extra credit. It due `only` on your final design. You do not need it on your initial.

In what way did you make your code shorter. List everything you did!

To optimize the code, I will focus on minimizing unnecessary global variables and implementing efficient buffer handling techniques to enhance input and output performance. Additionally, I will incorporate robust error handling mechanisms to improve the overall reliability of the program. These changes will collectively contribute to a more effective and efficient solution for file processing. The program's design and usage will align with these optimization strategies, ensuring a streamlined and efficient workflow for users.

References

[1] Wikipedia contributors. C (programming language) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/C_(programming_language), 2023. [Online; accessed 20-April-2023].

[2] Robert Mecklenburg. *Managing Projects with GNU Make, 3rd ed.* O'Reilly, Cambridge, Mass., 2005.

[3] Walter R. Tschinkel. Just scoring points. *The Chronicle of Higher Education*, 53(32):B13, 2007.