Assignment 3 – XD Report

Jackson C. Dawson

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.

The "XD" program replicates some of the functionality offered by XXD. The user provides XD with data, and is returned a "hex dump" of the contents of their data, buffered in 16-byte chunks. This includes an offset-marker that helps the user to understand where in the data each character exists. Each line (representing a 16-byte section of text) also contains the text in hexidecimal format, as well as its ASCII representation if applicable.

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 dedicated portion of memory that is used to "transport" data from one place to another. For example, you may read some data from location A into location B. This data may move at x mb/s, adding to the buffer, then being read from the buffer. This may be useful to avoid thread blocking. If you buffer content in smaller chunks, you are perhaps able to complete urgent tasks in between chunks of data transportation.

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

  The return value of read() is an integer (often represented using ssize_t). It will either return a positive integer- indicating the number of bytes that were read successfully, zero- indicating that the end of the file was reached, or a negative integer- indicating that an error occurred during the read operation. The inputs for read() are 1. a file descriptor, 2. a buffer, and 3. the count. The file descriptor is an integer which represents the location of data to read from; the buffer is a pointer to an area where read data will be stored; the count ("nbytes") is the amount of bytes that you wish to read at a time. The count is often represented using size_t.

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

  A file number (also called a file descriptor) is an integer which uniquely identifies an open file (presumably to read/write/append to). the file numbers for stdin, stdout, and stderr are 0, 1 and 2, respectively.

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

  When read(0,16) is called, read() will attempt to read from file descriptor '0' (stdin). 16 will be returned if the read operation successfully read 16 bytes, implying also that the data read contains at least 16 bytes. It will not return 16 if there are less than 16 bytes of data to read, if the process is interrupted for some reason (e.g. 'ctrl+C'), or if the process fails for any other reason.

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

  One example of such a situation might be attempting to read an extraordinarily large file. In fact, it just needs to be the case that the file that is being read is larger than the buffer. In this case, the reading process needs to be broken up into buffer-size chunks. Another situation may be getting a flow of data from an outside source (e.g. streaming video). It would be wildly impractical to read an entire video file for a user, as the user would have to wait for the process to complete, it opens a risk of the connection breaking, and it insists on using available resources that may ultimately not be worth it (e.g. user doesn't watch full video, or wants a different fidelity). In this instance, it would be much smarter and safer to read the video contents using a buffer.

Testing

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

As far as unit testing goes, I need to test printHexAndAscii() to ensure it prints the correct format for various buffer inputs. If I decide to break up my code into even smaller functions, I will need to test the functionality of those functions as well. Boundary testing is also needed. This entails examining edge cases like empty files or files whose sizes are multiples of the BUFFER_SIZE. Other input tests should include different file types, such as binary, text, and executables. Error handling should be implemented to ensure the appropriate failure response encountering nonexistent files, directories passed as files, or permission issues. Testing argument variety is essential as well. This should cover scenarios with no arguments, one argument, and multiple arguments.

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].

[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.

Start by compiling the program to make it usable. This is achieved by typing the following commands into the terminal:

```
make format
make
```

Great, now the program "XD" is ready to use. If you have a file with data you want to read, use the following command to read it using XD:[3]

```
./xd <your_file>
```

If you don't have data ready to read, you can still use XD. Instead of including a second argument when running XD, simply do:

```
./xd
```

Now, you will be able to paste or type anything you would like XD to parse, and execute the program using 'return':

```
<your input>\n
```

After each round of processing, the terminal will stay open for new inputs until you exit the program using:

```
^C
```

To clean up after yourself, do:

```
make clean
```

You're all done!

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.

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.

'XD' is a command-line utility designed to read a file or standard input and display its contents in both hexadecimal and ASCII format. The program breaks down into two main parts: reading input and output formatting.

The main function serves as the entry point and guides the overall program. It begins by handling command-line arguments, where it expects either zero or one argument. With zero arguments, it reads from standard input (STDIN), and with one argument, it attempts to open the specified file. If more than one argument is provided, or if the file cannot be opened, it returns 1, indicating an error.

When reading from a file or STDIN, the main function enters a loop where it reads up to BUFFER_SIZE (16) bytes at a time into a buffer. After each read, it calls the printRow function, passing the buffer, the number of bytes read, and the current offset within the stream.

The printRow function is responsible for the formatted output. It receives the buffer and the number of bytes to process. It prints the offset in hexadecimal, followed by the hexadecimal representation of the bytes. For bytes that represent printable ASCII characters, it includes them in an ASCII representation; for other bytes, it prints a dot ('.'). It ensures proper alignment and spacing of the output, arranging the hexadecimal bytes in two-character groups and capping the ASCII representation at 16 characters.

[3] Make sure to include the appropriate file extension! (e.g. '.txt')

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.**

int xd(file: int):
- Runs program
- argument "file" is optional

int main(int argc, char *argv[]):
- The entry point of the program. Its purpose is to process any input arguments, open the file (or use standard input), read the file's contents into a buffer, and print each row using the printRow function.

void printRow(const unsigned char *buffer, ssize_t bytesRead, off_t offset):
- The purpose of this function is to format and print a single row of the file's contents, showing the offset in the file, the hexadecimal values of the bytes, and their printable ASCII characters if they are representable as such.

1. Print the offset in hexadecimal format.
2. Initialize a character array to hold ASCII representations.
3. For each byte in the buffer:
   - If the byte was read from the file, print it in hexadecimal and add its ASCII character to the array.
   - If the byte was not read (end of file), print spaces and terminate the ASCII array.
   - Every two bytes, add a space for readability.
4. After processing all bytes, print the ASCII character array.

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!

Some changes are kind of trivial, but definitely helped to reduce the character count. I found two include's that I wasn't actually using a got rid of them. I also got rid of every comment and unneccesary new-line. I shortened every variable name to as small as possible (1 letter max). I turned if-else statements into ternary operator statements as much as I could. I changed *argv[] to **argv for that single character edge. I combined a check to return 1 from two if-then statements into one if-or-then statement. I got rid of as many brackets as possible. I changed from a while (1) loop to a for loop in my main function. I discovered that I could safely call ”close(fd)” without ever opening it, meaning i didnt have to check in order to close it or not. The last thing I wanted to do was take my print function and just put the code into the body of my main function (inline), but it was giving me a few issues that I didn’t care to sort out, so the function declaration and call is still there. Definitely not ideal but not worth fixing it either.

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.