Xuefeng Guan ID: 920016536
Github: XuefengGuan1 CSC415 Operating Systems

Assignment 5 – Buffered I/O

Description:
This assignment is to simulate how buffers work when a user is requesting data from a file using functions like open, read or close.

Approach:
An overall easier assignment compared to other assignments. Mainly test us on using functions written by someone else.

Although this assignment was one of the easier ones. I still spent a considerable amount of time doing this. The b_open function is straightforward to implement, just need to call b_getFCB, to get a file descriptor; and call the GetFileInfo to get the "low-level file information", which are the fileName, fileSize and the location of the file.

Once a user requests to open a file, then b_open will create a file descriptor of this file, and init the struct. Return the file descriptor to the user.

Then jumping to the b_read function, this is the "main part" of this assignment. After the user requests to get a certain byte of data from the file, we need to read the file and fill our buffer first, then pass in the users' required amount of data to the user's buffer. The trick in this part is that the system will only read a block(512 byte in this case) of data, so we have to read a block of data to our own buffer, then pass it to the user accordingly.

I have 3 steps in my b_read:

1. Check if there are any remaining bytes in our own buffer, if the buffer is empty then do nothing, if there are bytes in the buffer:
    a. Check if our buffer has enough bytes to give to the user, if there is, copy to the user, and then return
    b. If our buffer is not enough, then give the user everything in our buffer.
2. Now that we have emptied our buffer, we should check if the remaining count is larger than a block. Then directly LBAread a block of data to the user's buffer. This step ends when the remaining count is smaller than a block.
3. Our buffer should be empty now, so we LBAread a block to our buffer, and then copy the remaining count to the user.

The 3 steps are the overall structure of the read function. But there are a lot of small things to keep track of, which was confusing to me, and I spent a lot of time getting them right:

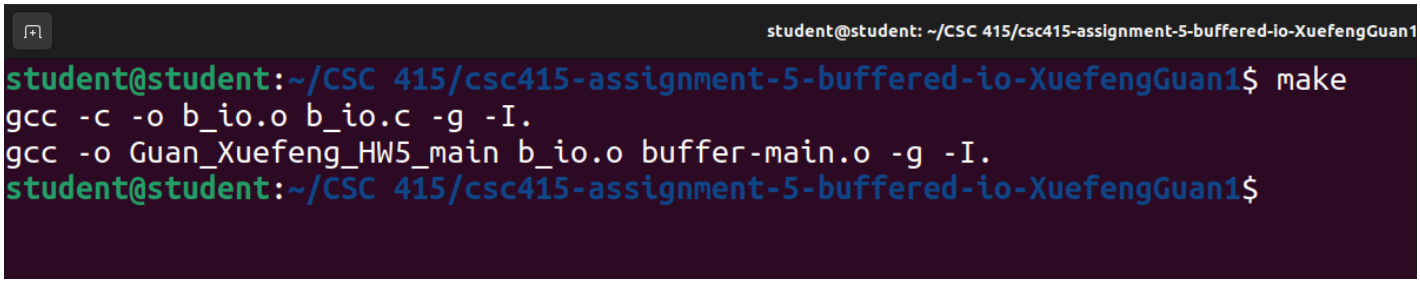
- I gotta make sure that when EOF is encountered, the program will remember that, and return 0 at the next iteration directly, without going deep into the program.
- I gotta make sure that when using memcpy, the third parameter which is the number of bytes to copy is not greater than our buffer's length, otherwise this would cause overflow.

- I gotta make sure to keep track of which index of the user's buffer and my buffer is on.

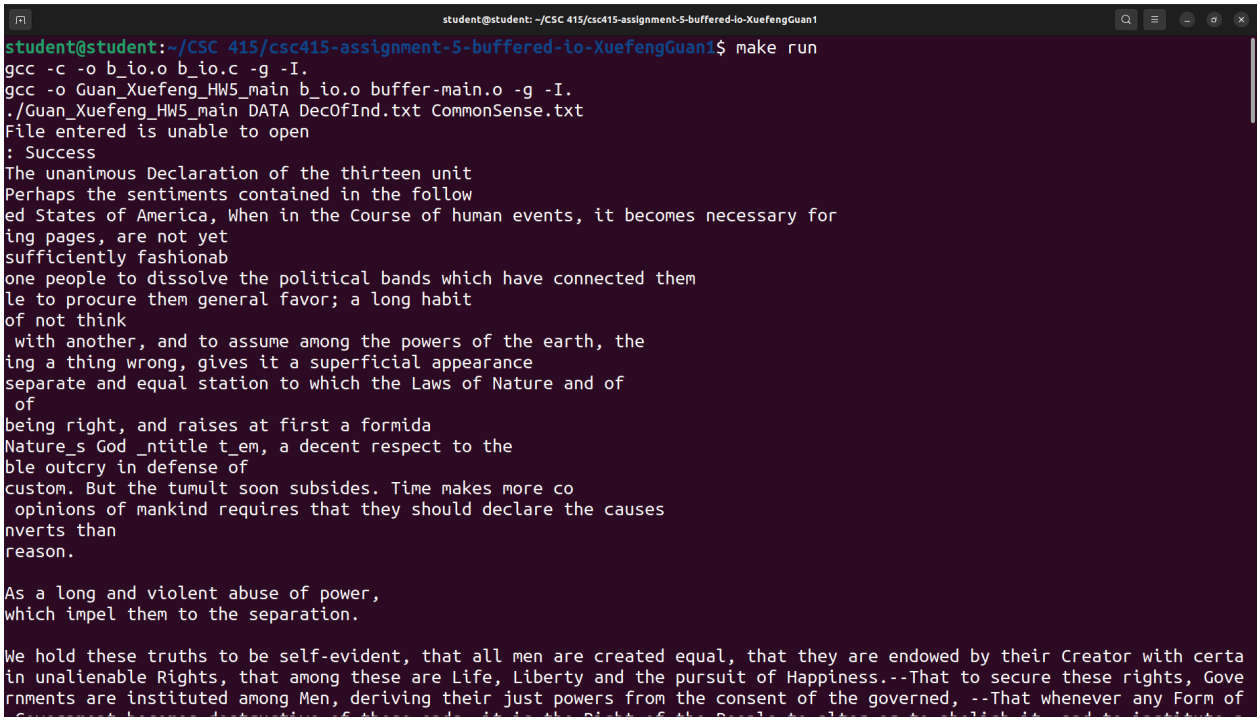
Issues and Resolutions:

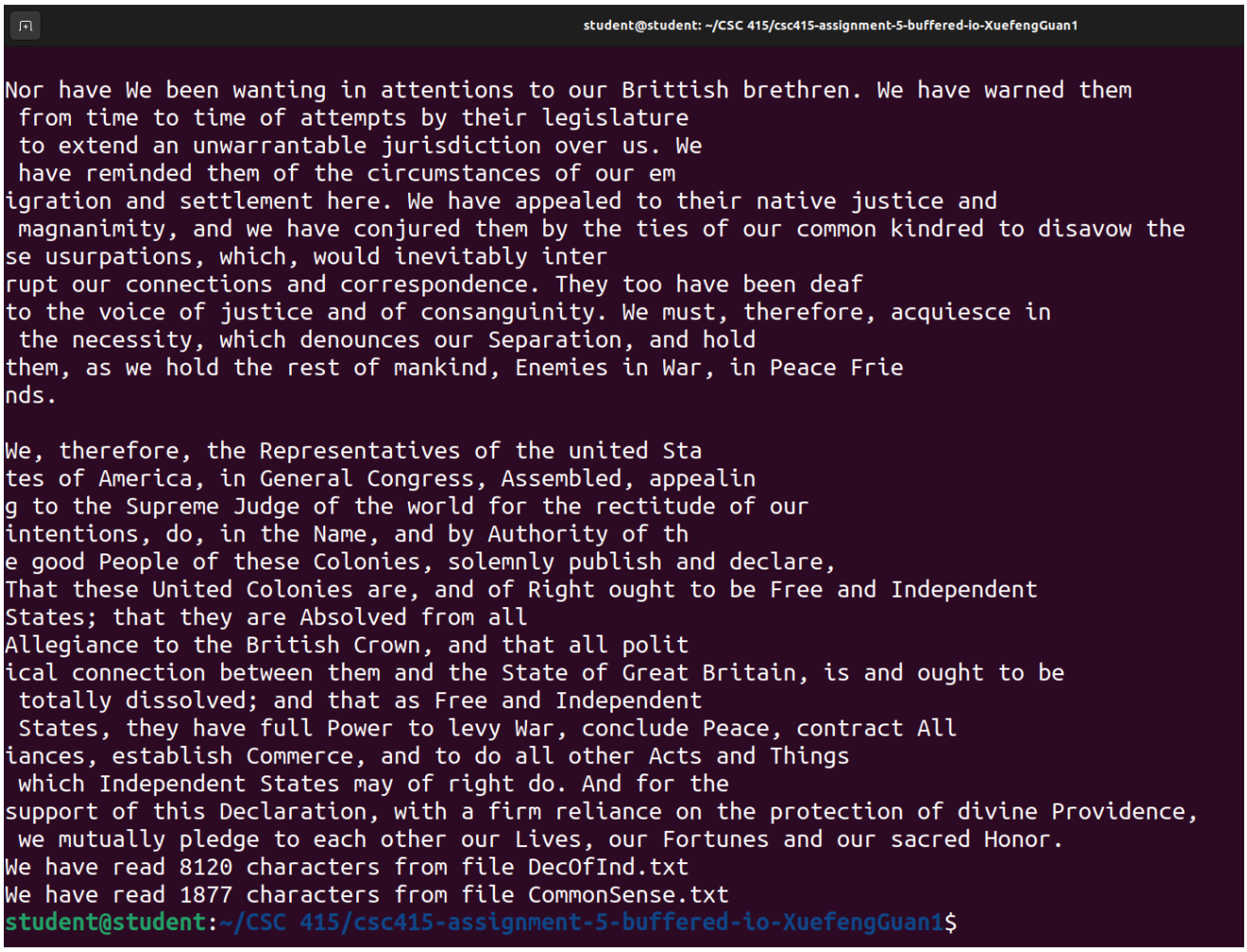
- The whole assignment I was trying to get the algorithm for the b_read straight, I kept getting weird answers. One thing bothered me the most was that I didn't know for example, if there are only 5 chars left in the file, and the count is 10, this means the user wants us to give them 10 more chars. When we do LBAread(myBuffer, 1, location); myBuffer will only have 5 chars. And then if I do memcpy(userBuffer, myBuffer, count). This would cause overflow, and there were no errors indiciatin me that. I spent hours trying to figure out why I'm getting weird results, then I realized it was just memcpy doing buffer overflow.

Screenshot of compilation:



Screen shot(s) of the execution of the program:





The output of this program is 8120 for DecOfInd.txt and 1877 for CommonSense.txt