Robel Ayelew ID:922419937
Github: RobelKasahun CSC415 Operating Systems

Assignment 5 – Buffered I/O

Assignment description:

This assignment is about Buffered I/O, meaning our role in this assignment is to act or be the system call functions such as open(), read(), and close(). Thus our job in this assignment is to handle input and output buffering by opening a given file, reading certain bytes from a file, as well as making the routine that would deallocate dynamically allocated resources that we utilized in the program.

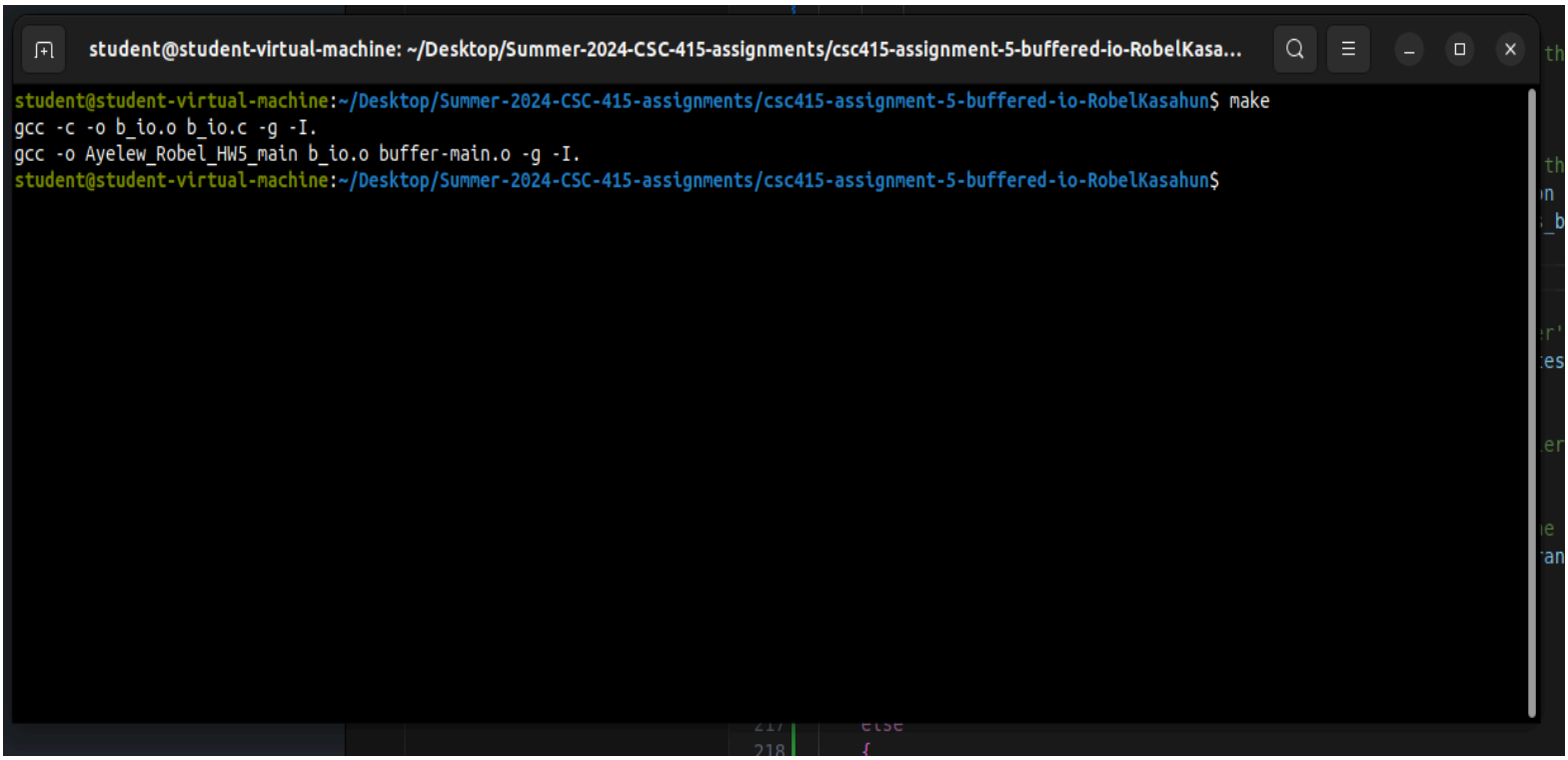
Assignment approach:

The assignment we were given initially was implemented therefore my duty is to add my logic to the routines or functions such as *b_open(), b_read(),* and *b_close()* so that these functions work as the system call functions that Linux provided. My approach to this assignment would be to first add variables such as *current_bytes_buffer_holding* and *buffer* to *b_fcb* structure because according to the assignment instruction each volume control block has to have its own buffer. After adding variables to *b_fcb,* I would head to the b_open() function to initialize the file descriptor by calling the *getFCB()*. What that would do for me is that, it will give me or return a free file control block or index of the file control block from the array. Next, I would call the *GetFileInfo()* routine by passing the file name to give me a file structure that would give me the name of the file, size in bytes, and the starting block of the file. Most importantly, the file control block that we receive from the *getFCB()* must be initialized, thus what I would do is that, I would get the free file control block from the *fcbArray* and initialize its variables. At this point, the *b_open()* will successfully initialize the free file control block and its buffer will only store 512 bytes. Then we return the index of the free file control block as a file descriptor and this file descriptor will be utilized by the *b_read()* to read a given number of bytes from a file opened by the *b_open().* Next would be to implement the *b_read()* function which reads some bytes of data from a given file descriptor to a caller's buffer. Thus, my initial approach in this function would be to return *count* when the *count* is less than or equal to zero because at this stage there are no bytes to read since the *count* is either zero or negative number, but if *count* is greater than zero, I would read 1 block at a time from a certain position from the file into my buffer and transfer appropriate bytes into the caller's buffer. While doing this, I would keep track of the number of bytes transferred from my buffer to the caller's buffer because that's what I would do after finishing transferring.

Issues and Resolutions:

I had a lot of issues with this assignment, but the issue that took me longer than expected was getting the reading or transferring certain data work from the LBAread to my buffer and using memcpy inside the b_read() function. Fortunately, after trying more even though the due date was more than a day, I was able to get the solution. The problem with LBAread was that I was passing zero for some reason for the LBA block position, but after I changed the zero to (file location + bytes the buffer of the current file descriptor holding) divided by B_CHINK_SIZE, and was able to get the right result. Last but not least, I have a problem that I still cannot fix. The problem is that GetFileInfo() is returning a NULL value even though I passed the file name. Therefore, I am submitting the assignment without fixing the bug.

Screenshot of compilation:



Screenshot(s) of the execution of the program: