Siarhei Pushkin ID:
Github: spushkin

Assignment 5 – Buffered I/O

Description:

For this assignment, we focused on implementing buffered I/O operations in C. The main objectives were understanding advanced buffering techniques, tracking information for multiple files, and understanding low-level file functionality. The aim was to prepare for more complex file system projects by mastering the handling of end-of-file conditions, memory management, and understanding existing code. The task required creating functions to open, read, and close files, with careful attention to buffering and tracking file states. This assignment was designed to improve our understanding of low-level file operations and develop efficient memory management and file handling skills.

Approach:

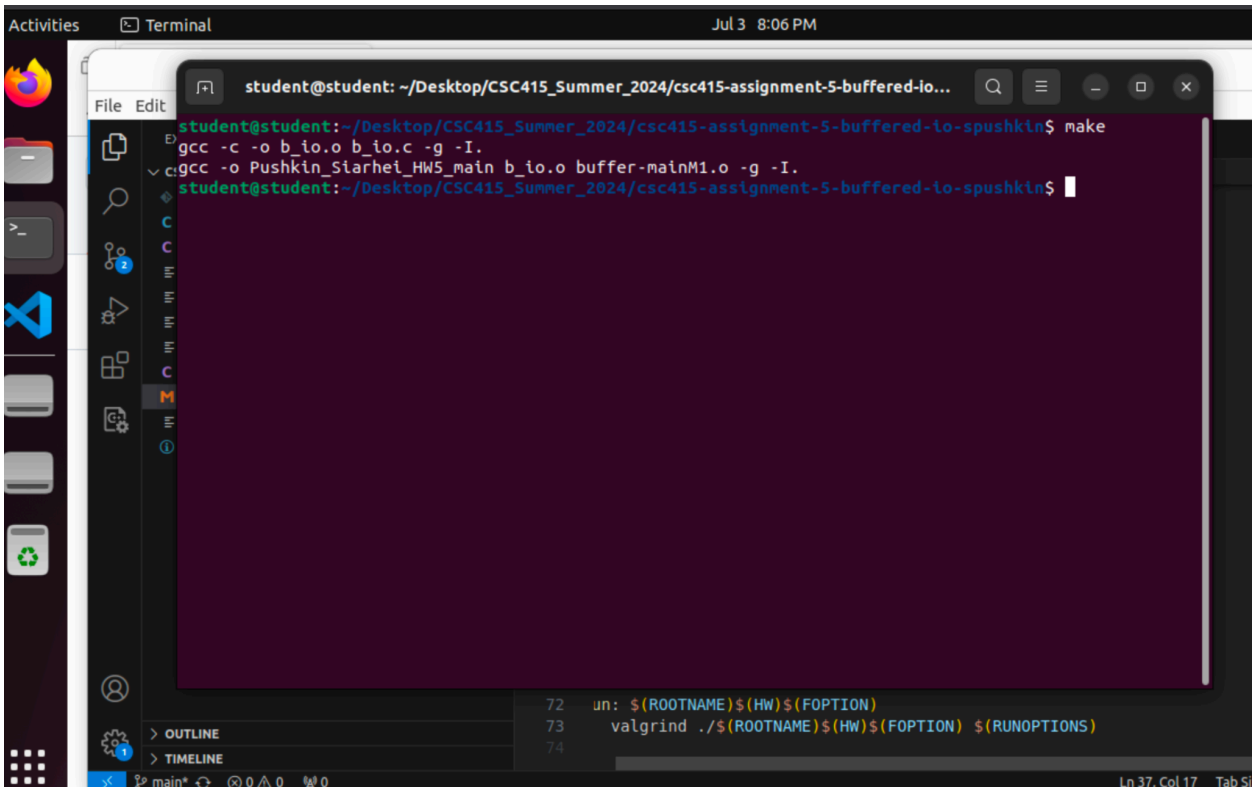
While working on this assignment, I implemented buffered I/O operations in C, prioritizing efficient file reading with fixed-size buffers. I began by thoroughly understanding the requirements and constraints, then examined the provided skeleton code to comprehend the program's structure and expected behavior. This initial analysis allowed me to grasp the assignment thoroughly and modify the provided code as needed. As required, I created three functions: `b_open,` `b_read,` and `b_close` in the `b_io.c` file, using the supplied low-level APIs (`LBAread` and `GetFileInfo`). I first implemented the `b_open` function, which entailed allocating a buffer for each open file and setting up the file descriptor and other necessary information. I used an array of structures to handle multiple open files, with each element representing a file's state. This method enabled me to keep track of each file's buffer, file descriptor, and other details separately, ensuring that file operations did not interfere. For the `b_read` function, I focused on reading data from the file into the buffer and transferring it to the caller's buffer in chunks of `B_CHUNK_SIZE` (512). I carefully managed scenarios where the requested read size exceeded the buffer size, ensuring accurate data reading and transfer. I also implemented logic to handle end-of-file conditions and track the remaining bytes to be read. In the `b_close` function, I ensured the proper release of all resources associated with the file, including freeing buffer memory and resetting the file state. Throughout the implementation, I tried to pay close attention to memory management to prevent memory leaks and ensure efficient resource utilization.

Issues and Resolutions:

One significant challenge I faced was proper management of the buffer during read operations to ensure seamless data transfer to the caller's buffer during read operations. This involved handling situations where the buffer was not completely filled or when the end of the file was reached. To address this, I carefully monitored the buffer's state, offset, and the number of bytes left to read, ensuring accurate data transfer and proper management of end-of-file scenarios. By overcoming these issues, I ensured the reliability of the read operations and

enhanced the program's overall stability. Overall, this assignment significantly improved my knowledge of buffering strategies, file operations, and efficient memory management in C, providing valuable insights into low-level I/O handling and careful resource management.

Screen shot of compilation:



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