**CSC415-Assignment-5**

**Assignment 5 – Buffered I/O**

Welcome to your next homework assignment.

Let me remind you of the expectations. Code should be neat, well documented. Variables should have meaningful names and be in a consistent format (I do not care if you use camelCase or under\_scores in variables but be consistent. In addition, each file should have a standard header as defined below. Remember your line length limits of generally 80 characters and no more than 100.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Class: CSC-415-0# Spring 2023

\* Name:

\* Student ID:

\* GitHub UserID:

\* Project: Assignment 5 – Buffered I/O read

\*

\* File: <name of this file>

\*

\* Description:

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

This is an INDIVIDUAL assignment. You can (and should) work in groups to research how to do the assignment, but each person must code their own version and make their own submission.

This assignment is to get you to handle buffered IO where you do the buffering.

You will see in this assignment that you may question why you are doing operations the way specified here in the assignment. This is because we are writing a buffering I/O. We will see later in file system project how important this is.

You are to create a set of routines in a file called b\_io.c, skeleton of this file is provided. The prototypes for these functions are in a file called b\_io.h (provided). The Makefile is supplied and you should set your FIRSTNAME and LASTNAME. (*make no other changes to the Makefile*)

You will be just writing three functions in b\_io.c

b\_io\_fd b\_open (char \* filename, int flags);

int b\_read (b\_io\_fd fd, char \* buffer, int count);

int b\_close (b\_io\_fd fd);

Your functions here will only use the supplied lowlevel APIs. i.e. LBAread, and GetFileInfo.

The b\_open should return a integer file descriptor (a number that you can track the file). You may want to also allocate the 512 (B\_CHUNK\_SIZE) byte buffer you will need for read operations here. Make sure though you know how to track the buffer for each individual file. Return a negative number if there is an error. You will call GetFileInfo to find the filesize and location of the desired file. See the structure fileInfo. GetFileInfo returns a pointer to fileInfo (this pointer does NOT need to be freed). That structure has the starting block number (all files are contiguously allocated) for the file and the files actual byte length. If the file is not found, GetFileInfo returns NULL.

The b\_read takes a file descriptor, a buffer and the number of bytes desired. The Operation of your b\_readfunction must only read B\_CHUNK\_SIZE byte chunks at a time from LBAread into your own buffer, you will then copy the appropriate bytes from your buffer to the caller’s buffer (do not copy one byte at a time and treat the data as binary data). This means you may not even need to do a read of the actual file if your buffer already has the data needed. Or, it may mean that you have some bytes in the buffer, but not enough and have to transfer what you have, read the next B\_CHUNK\_SIZE bytes, then copy the remaining needed bytes to the caller’s buffer.  
The return value is the number of bytes you have transferred to the caller’s buffer. When it is positive but less than the request, it means you have reached the end of file. Hint: You may also need to track between calls where in your buffer you left off, and which block of the file you are at.

You also need to be able to handle if the read request is greater than B\_CHUNK\_SIZE, meaning that you may have to directly fill the caller's buffer from a B\_CHUNK\_SIZE byte read (no need to buffer) then buffer just any amount needed to complete the caller's read request.

You are also responsible for keeping track of the file size, and once you reach the end of file, return 0 indicating there are no more bytes to read.

The b\_close should free any resources you were using.

You can write additional helper routines as needed.

Limits: You can assume no more than 20 files open at a time. (i.e. you need to ensure that multiple files can be open at one time, so the buffer you have for a file can not be global, but must be associated with that open file. A function to get an available File Control Block (FCB) is provided in b\_io.c.

The main program (buffer-main.o, provided) uses the command line arguments to specify data file and the desired target file(s). The main program uses b\_open, reads some variable number of characters at a time from the file using b\_read, prints those characters to the screen (ending in a newline character), and loops until it has read the entire file, then b\_close the file and exit. The program may open multiple files at one time and will request a chunk and print that one for each file, so the output will contain a line from file 1 than a line from file 2, etc until the end of file.

You are provided the file DATA as the file you are accessing and that contains the files IHaveADream.txt, CommonSense.txt, DecOfInd.txt. Think of that file as the "hard drive".

The output will vary each run but at the end you should see:

We have read 8120 characters from file DecOfInd.txt

We have read 1877 characters from file CommonSense.txt

You should submit your source code file(s) and Makefile, along with a writeup in PDF format using the writeup template that includes a description of what you did, show me the logic and steps you define to complete this project and the compilation and execution output from your program in GitHub, and the PDF also in Canvas. There is no Analysis section needed for this assignment.

The ONLY files you will modify is the Makefile and b\_io.c (you will not be creating any new files)

Rubric

| **Grade Item** | **Grade Points** |
| --- | --- |
| Standard Header | 2 |
| Proper open and malloc | 15 |
| Proper tracking and implementation of read | 20 |
| Proper buffering | 25 |
| Proper implementation of close and free | 5 |
| Correct Output | 10 |
| Code Comments | 10 |
| Writeup | 13 (Description, Compilation, Sample Output) |