Assignment #5 - Buffered I/O

- Assignment: Handle buffered I/O where you do the buffering
  - Create a set of routines in the file b_io.c
    - Prototypes are contained in b_io.h

    ↳ The functions will only use the supplied low level API's (LBAread, and GetFileInfo)

- b_open - Should return a integer file descriptor (a number that you can track the file)
  - ✓ · May want to allocate the 512 (B_CHUNK_SIZE) byte buffer for read operations
  - ✓ · 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 (contained in fsLowSmall.h)
  - ✓ · GetFileInfo returns a pointer to fileInfo (this pointer does NOT need to be freed)
    - The structure has the starting block number for the file and the files actual block length
  - · Error identified: Can't handle opening more than 9 files at the moment 7/7 04:01 AM
- b_read - Takes a file descriptor, a buffer and the number of bytes desired.
  - ✓ · The operation of your b_read function must only read B_CHUNK_SIZE bytes 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. Treat the data as binary data)
    This means you may not even need to do 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 the file.
    - HINT: You may need to track between calls where in your buffer you left off, and which block of your file you are at.
  - ✓ · 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 responsible for keeping track of the file size, and once you reach the end of the file, return 0 indicating there are no more bytes to read.

- b_close - Should free any resources you are using
- You can write any additional helper routines as needed.

- Limits
  - You can assume no more than 20 files open at a time ✓ b_open
    - Assuming multiple files can be open at one time, this means the buffer you have for a file can not be global, but must be associated with that open file. (b_getFCB) A function to get an available FCB is provided in b_io.c

- How it works)
  - 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)
  - loops until it has read the entire file
  - b_close the file and exit

IHaveADream.txt - 1521 bytes
CommonSense.txt - 1877 bytes

Lecture Review: File System is a block operation - Learn to read and write blocks
- Implement versions of the open, read, close functions used in assignment 4

b_open: Return a file descriptor (Index into an array of file control blocks)
- May want to allocate buffer here
- Know how to track a buffer for each individual file

b_read: Takes file descriptor, buffer and number of bytes desired.
Example: Read 10 bytes with nothing in buffer
- Read the first block of the file (LBAread(buffer, 1, x))
  - This loads the data into the buffer
- Take the 10 bytes and copy it into their buffer
  - You will then return - Index is at 10 of the current file

Read 500 bytes
- Take the next 500 bytes and copy it into their buffer
- Buffer should be at 510
- You will then return

Read 10 more bytes
- Take the next 2 bytes and copy it into their buffer
- Read the next block of data - this overwrites everything
- Position should now be at 0
  Take the next 8 bytes to reach the 10 bytes requests
- You will then return

Read 200 more bytes - but file is only 600 bytes so only 80 left to read
- You must be able to detect the end of file
  - If you give them less than asked, it indicates the end of file
- Return number of bytes put into buffer (0 if none put in)

Hint for end of file: There is an easy way?
LBAread is the only way to get data into your buffer (No linux open or read)
- Hint: You may also need to track between calls where you are in the buffer like assignment 2

- How do we find a file?
    - Call getFileInfo to find the fileSize and location. Returns a pointer. (Does not need to be freed) The structure has a starting block number (all files are contiguously allocated) for the file and the files actual byte length.

- Your responsibility of keeping track of the file size, and returning 0 indicating there are no more bytes to read.

- 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)
- Loops until it reads the entire file
- Calls 'b_close' and exits
- Every run will be different since it pulls a random character to read each time

Looking at files: fsLowDs

- getFileInfo - NULL if can't open - Returns pointer with info
    - Given n, returns number of blocks

Calculating blocks
        - int getBlocks (int length of file)
          int n = (length of File / 512) + 1 $(m + n - 1) / n$
          3 = 1024 / 512 + 1

- LBAread - takes the pointer, takes how many blocks you want to read, and which block #

b_io.h
- Prototypes

b_io.c
- Put the buffer in b_fcb struct - Probably pointer to not waste space
- b_open:
    - Call getFileInfo and b_getFCB
- b_read:
    - Take fd to open, Takes a user buffer, Takes how many bytes the user wants to rea
- b_close
    - Close the resources

b_read: Default case - memcpy into user buffer
change buffer position
return bytes committed to buffer (in this case count

w/ Count : 11394
2773

w/ actual Count : 11363
2688

if (

LBA read: Returns '1' if successful load
Returns '0' if block number is invalid? Is this end of file?
- Means nothing new was loaded into buffer - End of file reached

totalCount :
count : 70
textLeft = 28
currentPos = 484

```
      28        70
while ( textLeft <= tempLeft )
  memcpy ( 28 )
  tempLeft = 70 - 28 = 42
  currentPos = 0
  textLeft = 15
      15       42
  while ( textLeft < tempLeft )
    memcpy ( 15 )
    tempLeft = 42 - 15
    currentPos =
    textLeft =

+ else ( textLeft > tempLeft )
  memcpy
  currentPos =
  textLeft =
```

28
+
15