Project 3: FAT32 File System Implementation

Project description -
Segmentation Fault: Core Dumped

Since Last Time…

- Knew how to implement basic commands manipulating the FAT32 image file
- Understood how to traverse the filesystem stored inside FAT32
- Did everything you can regarding directories (list them, change directories, remove and make new directories)

So What Next?

- Now we will look at file based operations in FAT32 image
- Basically 4 major operations, classified in 2 categories
- Read Based - open, close, read
- Write Based - write

Some Things To Note

- Use rb+ as the mode of opening the image file
- Modifying the file image is risky. You may overwrite/ corrupt the file system
- Easy way to check that is running ls/cd commands just after that to ensure everything is in order
- Use the hex editor to check whether correct values are placed in the correct places

Modes: Concept

- A file can have basically 3 types of modes: Read Only, Write Only, and Read and Write

- In case of this project, read only means no editing of the file (you can only use read command when it is open), write only means the opposite (you can only use write command when it is open)

- Read and Write means both of these operations can be done

- Write and Read is the same

- You can make different interpretations of the RW and WR modes, as long as your explanation is consistent, it will be fine

Open and Closing Files

- How do you know which files are open and which one is not?
- Make a linked list / array / something that can hold the information about open files
- Basically this linked list maintains the list of files that are currently open
- Make a struct { int file_first_cluster_number; short int mode (or whichever method you choose for mode)}
- Make an array/linked list of the struct
- How do you use short int for mode? Simple.
- Use short int for mode. =1 is read, 2 is write, 3 is rw/wr

open FILENAME MODE

- Check if FILENAME is present in current directory
- Check if FILENAME is not a directory (DIR_Attr&0x10 == 0x00)
- Check if FILENAME has the proper mode permission. For example, if FILENAME is READ_ONLY, then, DIR_Attr&0x01 == 0x01). Else, it is not marked as READ_ONLY (you can read as well as write it).
- If yes, then get the first_cluster_number of the FILENAME from its Directory Entry (remember that filenames are in directory entries. Example, look at FATSpec.pdf in the root directory. Thats a file. It has a directory entry too)
- Check if FILENAME's first_cluster_number is in the linked list or not (the linked list basically is your list of files that are open) If the first_cluster_number is present inside the linked list already, then it is already open. Hence print error
- If FILENAME is READ_ONLY and MODE is also READ_ONLY, then append the linkedlist/array with the FILENAME's first_cluster_number and the mode of the file
- If FILENAME is not READ_ONLY, immediately append the linked list/array with the FILENAME's first_cluster_number and the MODE of the file (as the mode doesn't matter, we actually have RW or W or R mode here)
- Otherwise, the file is in READ_ONLY mode and is being opened in RW or W mode. Or it is a directory. Or it is not present. Hence print error

close FILENAME

- Check if FILENAME is in current directory (to get the first_cluster_number)
- No need to check if FILENAME are opened in right mode (you did that while opening remember)?
- Check if linkedlist/array has the first_cluster_number of FILENAME in it.
- Delete that entry

Now we go to reading files

- But before we do go to reading files, we need some concept revision

- Basically what is the difference between directories and files in terms of storage?

- Do files have a separate data structure like directories?

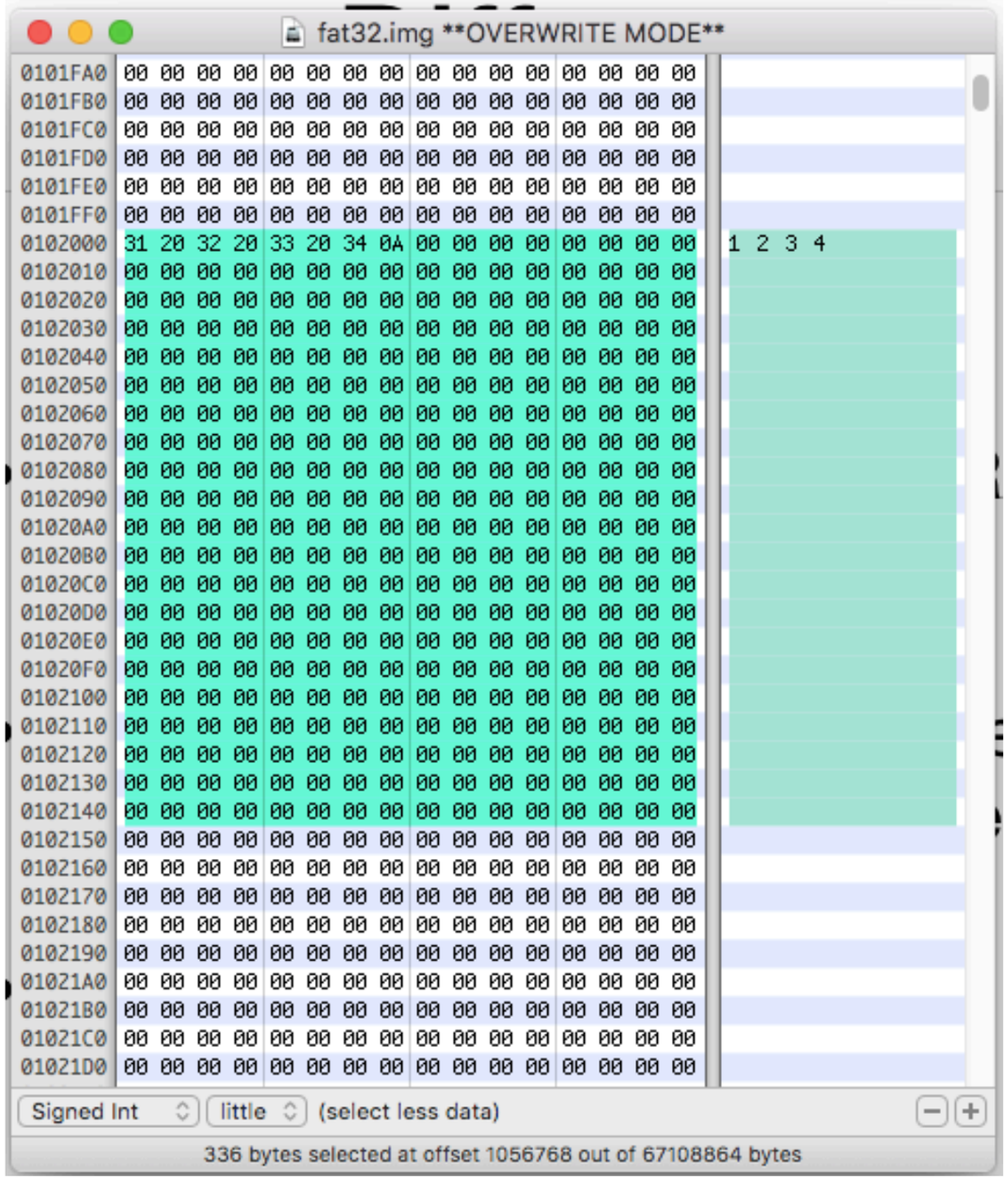
Directory and File Structure Differences in FAT32

- The data in a file is stored in RAW format inside the FAT32 image file

- This implies that basically to read the file, you need to print the whole data inside the file as string

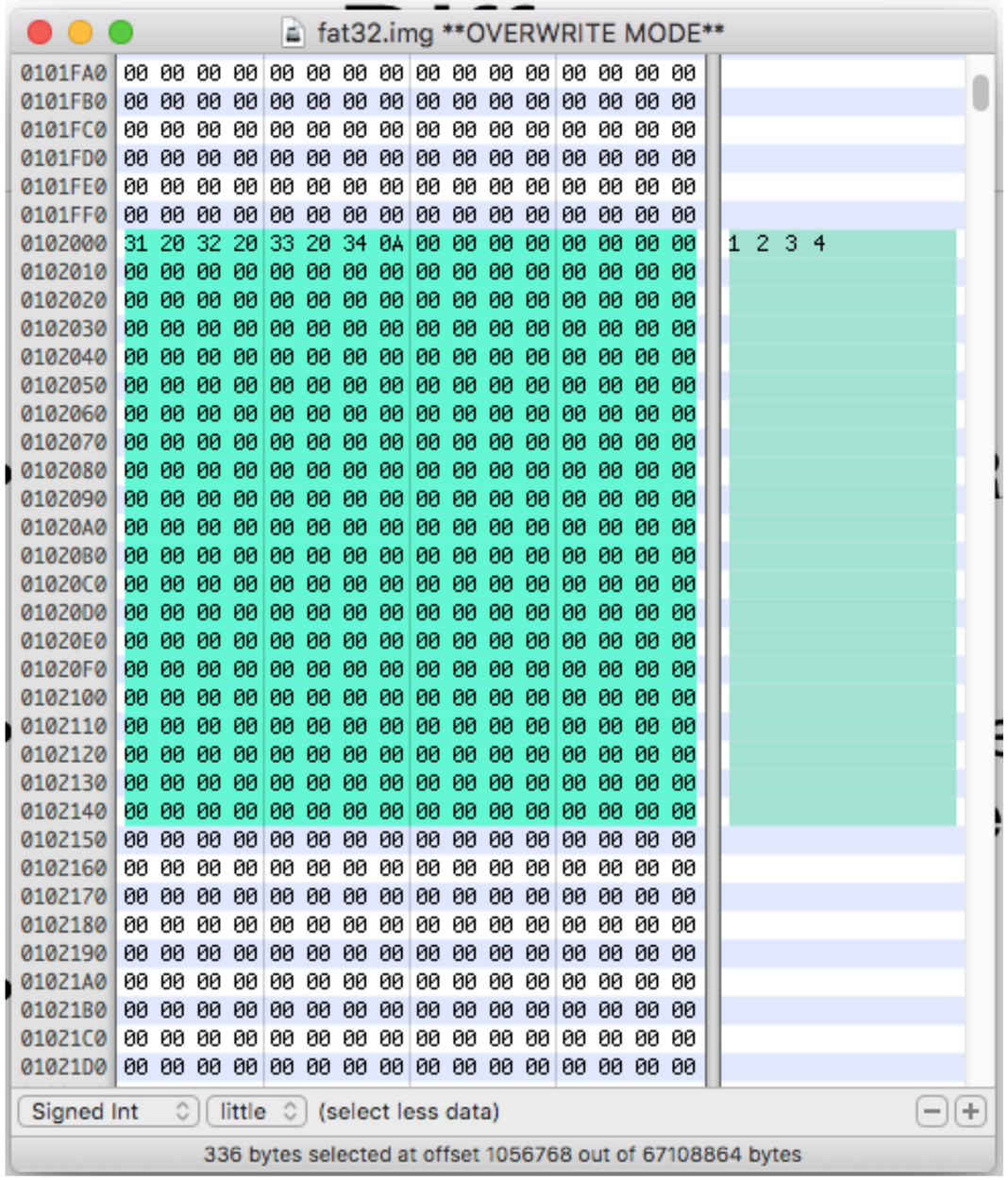
- Look at the example on your right!

- That is LITERALLY a text file that has “1 2 3 4” as its contents



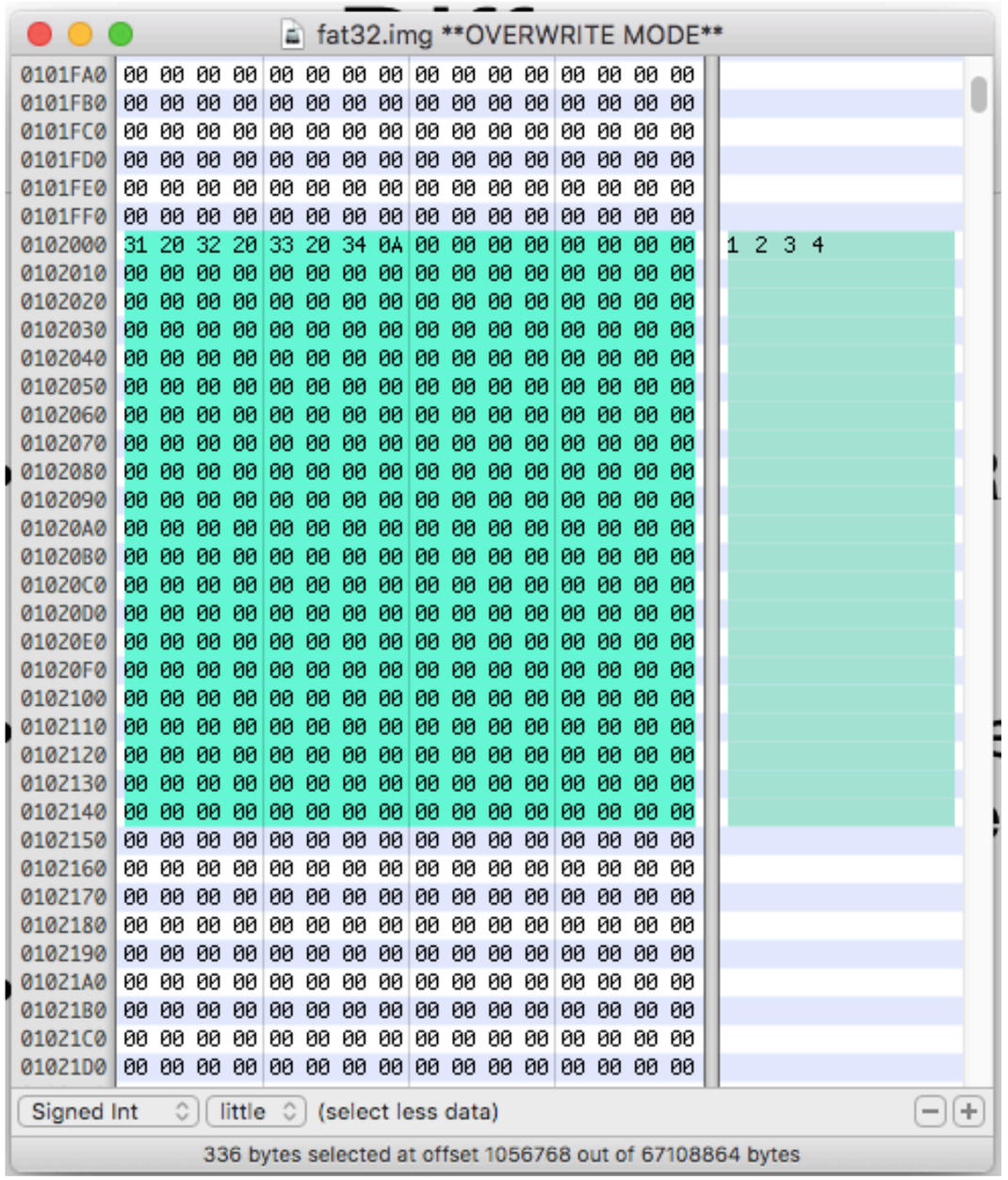
Directory and File Structure Differences in FAT32

- Hence, just print the whole cluster as a string.
- Make an char array with a size of sector_size (use malloc)
- fread the data from the cluster into the char array
- Then use puts(char array);



Reading the whole content of a file

- The same as ls.
- Go to the first_cluster_number of the FILENAME
- Make sure it is open (check the linked list and make sure it is in ‘r’ or ‘rw’ or ‘wr’(same as ‘rw’) mode)
- Then do the same as ls
- Difference is, instead of reading the directory entries, you read the whole cluster and print it out
- Then print the data in the next cluster , and so on till you reach end of file



read FILENAME OFFSET SIZE

- Problem statement:
- Given a FILENAME, read SIZE bytes of the FILENAME starting at OFFSET
- Problems: Edge cases such as - OFFSET > sizeof(FILENAME), SIZE > sizeof(FILENAME), or OFFSET + SIZE > sizeof(FILENAME)
- Project description as gospel, the first case prints error, second and third case prints the whole file data and sizeof(FILENAME) - OFFSET bytes of the file data respectively
- Hence, we will have to make some modifications in the read function that we have right now.
- OFFSET may not be perfectly divisible by sector_size (i.e. OFFSET may not be synchronized with the cluster. Eg. OFFSET may be 0x00001289 where sector_size is always 0x00000200. Hence OFFSET/SIZE = 0x00000009 (probably) and OFFSET%SIZE = 0x00000089.
- This means, in the above example, we will go to the 9th cluster of the file, starting from its 1st cluster, and then inside the file's 9th cluster, go to the 0x00000089th byte and start reading SIZE amount of bytes from there
- Hence allocate a char array of SIZE elements (char has 1 bytes remember) and then start doing this -

read FILENAME OFFSET SIZE

- Take in the OFFSET, divide it with sector_size, let the quotient be n (i.e. n = OFFSET/sector_size)
- This gives you the nth cluster from which you want to read from
- Then, while n != 0: go to next cluster
- while (n != 0){
  cluster_number = FAT[cluster_number];
  n-=1;}
- If cluster_number is 0x0000000 or 0xFFFFFFF8, print error
- Else, you are in the nth cluster which you need to read
- Now account for the rest of offset. fseek to (nth cluster's starting point + (OFFSET%sector_size))
- Then, fread data from this point unto SIZE bytes and print it out. If some part of the data is in another cluster, move to that cluster and print that rest part of the data too
- Edge cases - What if OFFSET < size of FILENAME, SIZE < size of FILENAME, but OFFSET+SIZE > FILENAME?
- Solution, check DIR_Size - OFFSET - SIZE >= 0, (implies OFFSET+SIZE < FILENAME), then do the things you did above
- Else, just print DIR_Size - OFFSET bytes of the FILENAME (Considering OFFSET < DIR_Size)

write FILENAME OFFSET SIZE STRING

- Now this is where it gets more complicated
- Just do the same as read till you start reading (go to the nth cluster, then the OFFSET%sector_size bytes since the start of the nth cluster)
- Now, ideally, you should fwrite the STRING into the FILE
- Initialize a char array of SIZE bytes
- Make sure to check that FILENAME is OPEN in WR_ONLY or RD_WR mode
- However, there are a bunch of edge cases that can happen:

write FILENAME OFFSET SIZE STRING

- What if we have a large file (say 1024 bytes, and you are supposed to write 20 bytes at OFFSET 100. What happens to the rest of the file?
    - Solution, overwrite the 20 bytes since OFFSET 100, and then 0 out the rest of the contents of the file.
    - If there are other clusters after the nth cluster for this file, unlink them (like you did in rm and rmdir).
    - Update the DIR_Size to the new size of FILENAME (n*sector_size + OFFSET%sector_size + SIZE)
- Another edge case is if the OFFSET is > sizeof(FILENAME)?
    - Solution: print error
- ANOTHER edge case is if OFFSET is < sizeof(FILENAME) but OFFSET+SIZE is > sizeof(FILENAME)?
- Solution, write the STRING on the (OFFSET/sector_size) cluster at OFFSET%sector_size point of the cluster.
    - If there is anything written there, overwrite.
    - However, as the STRING will increase the size of FILENAME, remember you may need to allocate a new cluster to the FILENAME. Also, put the correct value for DIR_Size at the end. (OFFSET+SIZE) should be the correct value now
- What if STRING is not equal to SIZE?
    - Well if STRING < SIZE, pad 0s (ASCII value 0) for the rest of the char array (char array should look like this = [’S’, ’T’, ‘R’, ‘I’, ’N’, ‘G’, 0x0 0x0 0x0 … till the end of the array],
    - Remember that the char array is initialized to have a size of SIZE value, hence the padding will be done for the last SIZE - sizeof(STRING) bytes
    - If STRING > SIZE, just take in the first SIZE bytes of the string and put them in the array. Write them to the file

write FILENAME OFFSET SIZE STRING

- Remember, writing to files are complicated. You may have to remove clusters like you did in rm/rmdir
- Or you may have to allocate new clusters to the file like you did in creat/mkdir
- Remember to update the directory entry of FILENAME using the new DIR_Size
- sizeof(OFFSET + SIZE) = new DIR_Size
- A good strategy should be to first write to a blank file
  Like creat abc, write abc 0 4 “ABCD”
- Once that works, move towards appending the file
  Like write abc 4, 4 “EFGH”
- Now start playing around with this, (if you do this previous step for quite sometime, you will need to allocate a new cluster)
- Then Start on overwriting the file. Like write abc 4, 4 “PQRS”

Project 3: Final Steps

- There are 15 unassigned people in Canvas. Please mail the group names BY THIS SUNDAY.
- If I don’t get the group members by this sunday, the groups will be assigned at random.
- To all -> Start finishing this project now
- Next week will be extended office hours. My Office Hours are from 1:30 PM - 4:30 PM at Majors Lab every Friday

MAY THE CODE

BE WITH YOU