# Lab 5 Solid-State Disk and File System

|  |  |
| --- | --- |
| **Goals** | * Create a write-once file system |
| **Review** | * Lecture 5 * Lab 5 starter files are attached. |
| **Starter files** |

**Background**

In this lab we will create a file system in the flash memory of the tm4c123. Flash memory is used because it will store data across system restarts and power-offs. This lab does not use the RTOS we have been developing over the last four labs and instead is purely focused on creating a filesystem to teach you the fundamentals of a file-allocation table and how segments of a file are stored. To simplify the lab we have provided you with the FlashProgram.c file which is used to interact with the low-level flash system in the form of memory blocks or sectors.

* In this lab we work with 256 blocks of 512 bytes.
  + Blocks 0-254 store data and block 255 is split in half to store the directory information and the file allocation table (FAT).
* We have simplified the lab to using only one directory which will contain all of our files. In this filesystem, there can be 255 files each occupying one block of data or 1 file occupying all 255 blocks of data, or any other distribution, but it is worth noting that we can have at most 255 files and 255 blocks.

The Directory data is organized by file numbers, and in the directory array Directory[i] points to the first block of data used by file i. If we try to read a file, like file 4, and we check Directory[4] and find it equals 255 then we will know that file does not exist. Directory data is stored in bytes 0-255 of the block 255. The FAT stores the location of the next subsequent block of data in a file, so file 1 may start on block 1 and FAT[1] = 2 so the second block of data is stored in the second block. Files have a tendency to get edited sporadically and thus won’t always be in order, so FAT[2] might = 17 so the third block of the first file is at block 17. This is called fragmentation and, while generally it slows the speed of reading files, we will not be trying to handle fragmentation in this lab. Finally, lets say we try to find the next block of data for file 1 and we find that FAT[17] = 255, this means that 17 is the last block in the file. The FAT is located in bytes 256-511 of block 255.

In more advanced file systems users can add data in the middle of a file and the data will get inserted between the old data, but in this lab, we will limit the filesystem to only allow us to add on to the end, or append, data to files. We will also not be deleting files or modifying old sections of data in any way. All we are worried about for now is creating files, allocating empty space to files, writing to the flash memory, and appending files.

# Preparation (do this before your lab period)

1: Go over the FlashProgram.c and understand how to use the different functions inside, these are important and will be necessary in creating your filesystem.

2: Read the relevant sections of your lecture to prepare for this lab.

# Procedure (do this during your lab period)

1. Implement the functions in eDisk.c
   1. eDisk\_ReadSector should read a sector from the flash memory into a buffer. This should be just like copying data from any array as the tm4c123 treats the flash and ram the same.
   2. eDisk\_WriteSector should write a sector to the flash memory. While the comments say that you can use Flash\_WriteArray or Flash\_FastWrite we want you to use Flash\_FastWrite as it is faster and requires more understanding to implement.
   3. eDisk\_Format should fill the addresses from EDISK\_ADDR\_MIN to EDISK\_ADDR\_MAX with 1s to indicate the drive is empty.
2. Implement the functions in eFile.c
   1. MountDirectory should read block 255 from the flash memory and fill out the directory and FAT arrays in RAM.
   2. lastsector should cycle through a file’s FAT and find the last block of data used for that file (it should have 255 stored in the FAT). We wan you to overcome the looping forever error by preventing the system from running more than 255 times (As there can’t be more than 255 files at once)
   3. findfreesector should attempt to find the last block of any file and return the block following that. For example, if file 1 ends at block 13, file 2 ends at block 11, and file 3 ends at block 18 then findfreesector should return 19 to give the next free sector. To make this easier I recommend taking advantage of the lastsector function you already wrote.
   4. appendfat should change the FAT entry of the last block of file “num” to “n”. For example, if file 3 ends at block 18 but we want to append block 21 to that file then you should change FAT[18] to 21; I would recommend using lastsector to simplify this process.
   5. OS\_File\_New should go through the Directory array and find a file that is empty (Directory[i] == 255). You should check to see if the directory is mounted before doing this and mount it if it isn’t already.
   6. OS\_File\_Size will work a lot like lastsector but instead of telling you where the last sector is it will count the number of sectors in the file.
   7. OS\_File\_Append should find a free sector, fill it with the array “buf” and append the FAT for file ”num”. I recommend using the functions you have already written to accomplish this.
   8. OS\_File\_Read should read the data located at a certain sector within a file. Let’s say file 1 has sectors 1->2->3->5->9, and we ask for the fourth sector of data in file 1, we should read and return the data in sector 5. You should also make sure not to go over the number of sectors in a file. Be sure to take advantage of the other functions you have written to do this.
   9. OS\_File\_Flush should recombine the Directory and FAT arrays into a 512 element array and write that to the flash memory. eDisk\_WriteSector should be useful here. Also note that you should not do this unless a directory is mounted.
   10. OS\_File\_Format should unmount the directory and format the disk using eDisk\_Format.
3. Now you should be able to run the lab and see the files on the left side of the screen and the FAT on the right. Pressing the buttons should allow you to navigate the tables to check for any errors.

# Checkout (show this to the TA)

1. Demonstrate the final system to the TA.

# Deliverables (exact components of the lab report)

* 1. Each team must submit a simple report to the Black Board. It needs to answer the following questions:
     1. What engineering issues did you find surrounding this File System? What are your suggested solutions?
     2. What advantages and disadvantages do this allocation method offer compared to other methods?
  2. Each lab member should submit a copy of the code to the lab 5 assignment under assignments on blackboard. Preferred submittal is in the form of a zip/rar/7z file.

# Hints

1. I will add down here again that you should attempt to reuse any functions you have already written in this lab instead of rewriting redundant code. It should make your code easier to read and write.
2. Once the lab is working you can erase the data by holding down the bottom button while restarting the board to make the tables easier to go through.
3. You may need to hit “erase flash” in keil under flash since the flash persists and can cause issues if you messed up somewhere and now the issue wont go away even after you have fixed the code.