COMP 304 Project 3

KU File System
Corresponding TA: Muhammad Aditya Sasongko
Due: June 1st, 2018

Notes: The project can be done **individually or teams of 2**. You may discuss the problems with other teams and post questions to the OS discussion forum but the submitted work must be your own work. This assignment is worth 9% of your total grade. START EARLY.

Any material you use from web should be properly cited in your report. Any sort of cheating will be harshly PUNISHED.

1 Description

In this project, you will implement a file system library which operates on a virtual hard disk. The file system is based on File Allocation Table (FAT), and it consists of only a single directory. The virtual hard disk which you will implement will be in the form of a single file. The file which functions as the virtual hard disk will have the following specifications.

- The virtual hard disk is divided into multiple fixed-sized regions called blocks.
- The number of blocks in the hard disk is fixed.
- Each block_size is 1024 bytes.
- update: The first block is the master record which contains the File Allocation Table. In its simplest form, the table contains file names, file sizes, and the ids or the offsets allocated to each file. Content of this block can be as follow.

  file1.txt 3000 2 3 4 file2.txt 3800 1 5 6 7

  In the above example, there are two files, i.e. file1.txt and file2.txt. The size of file1.txt is 3000 bytes and the ids of blocks allocated to it are 2, 3, and 4. The size of file2.txt is 3800 bytes and the ids of blocks allocated to it are 1, 5, 6, and 7.
- Each file contained in the virtual hard disk is assigned to at least a single block.
- A block is associated only to a single file, so multiple files cannot share a block.

2 File System Manipulation

To implement the file system library, you are expected to support following functions.

- (10 pts) **int kufs_create_disk(char* disk_name, int disk_size)** creates a file named disk_name with size disk_size * block_size. This function returns 0 if runs successfully, and -1 if it fails.
- (10 pts) **int kufs_mount(char* disk_name)** mounts the file system into memory. When this function is called, a file in current directory named disk_name, which represents the virtual hard disk, is opened. The first block in the hard disk will be read

and the content in it will be placed in a data structure which represents the file allocation table (FAT) that is loaded to the memory. update: The data structure should be designed so that it can contain entries of all files in the virtual hard disk, and each entry which corresponds to a file should contain the file name, file size, and the block ids allocated to the file.

Aside from the data structure, two global arrays should be initialized. One keeps track of the unallocated blocks in hard disk and the other contains the open file information (indices of files in the FAT data structure that are currently opened).

- (10 pts) **int kufs_umount()** unmounts the currently mounted virtual hard disk. When unmount happens, the following steps are taken.
  - The arrays of unallocated blocks and opened files are freed.
  - The first block in the hard disk is overwritten by the content of FAT data structure in memory. That means changes in the file system are only reflected to the hard disk when it is unmounted.
  - The FAT data structure in memory is freed.
- (10 pts) **int kufs_create(char* filename)** creates a new entry in FAT data structure for a file named filename. Before the entry is added, it is necessary to check if a file with similar name already exists in the FAT data structure. However, upon file creation no block is allocated to the file.
- (5 pts) **int kufs_open(char* filename)** returns the index of entry corresponding to the file named filename in the FAT data structure. This returned value is called file descriptor. Furthermore, the index will also be added as an entry in the opened file table.
- (5 pts) **int kufs_close(int fd)** update: deletes the entry in opened file table corresponding to file pointed by file descriptor fd.
- (5 pts) **int kufs_delete(char * filename)** When this function is called, the entry in FAT data structure corresponding to file named filename is deleted and blocks allocated for that file are marked as unallocated. However, the content of the file in the virtual hard disk doesn't need to be deleted. File deletion here means that the content in hard disk which used to belong to the deleted file is free to be over written.
- (10 pts) **int kufs_write(int fd, void* buf, int n)** This function writes to the file corresponding to file descriptor fd. The bytes to be written are from variable buf and the amount of bytes written is n. If the file has just been opened, the write will start from the beginning of the file. If the file has been written before after being opened, the written bytes are appended to the existing content. It means that there must be a variable which points to the offset of next write or read operation. This variable can be called file_position. You can implement this variable as an additional attribute in opened file table. update: After writing to a file and expanding the value of file_position, you need to check if the file_position becomes larger than the file size. In case it is larger than file size, you need to set the value of file size to the current file position. You can implement this variable as an additional attribute in opened file table. If blocks allocated to the file are full, a new block is allocated to the file. This function returns the amount of bytes successfully written.

- (10 pts) **int kufs_read(int fd, void* buf, int n)** reads the content of a file corresponding to file descriptor (index in FAT) fd starting from the file_position. The amount of bytes read from the file is n and the read bytes are written to buf variable. This function cannot read from regions beyond the file size. If this function starts reading from a byte position before the end of file, but continues reading until surpassing the end of file, the bytes that could be read by this function are only those before the end of file. In the end, this function returns the number of bytes read.
- (10 pts) **int kufs_seek(int fd, int n)** When this function is called, file_position is placed after the n^{th} byte in the file. If n is bigger than the file size, the file size is returned.
- (10 pts) **void kufs_dump_fat()** dumps the content of FAT data structure. The example is as follow.
  a.txt: 2 4 6
  b.txt: 1 3 5 7 8
  Each line in the example above corresponds to the name of a file in hard disk and the ids of blocks allocated to it. This function can be used for debugging purposes.

3 Provided Codes

- We will be providing an application to test your program. You can yourself develop such applications to test further functionalities of your file system.

4 Deliverables

You are required to submit the followings packed in a zip file (named your-username(s).zip) to blackboard :

- .c file that implements the file system. Please comment your implementation.
- any supplementary files for your implementation (e.g. Makefile)
- **(5 pts)** a README file briefly describing your implementation, particularly which parts of your code work.
- NOTE that there will NOT be a DEMO for this project so document your code and readme well.

GOOD LUCK.