CS342 Fall 2015 – Project 5
Very Simple File System (VSFS)

Last Update: Dec 12, 2015, 23:31

Assigned: November 12, 2015, Friday
Due date: December 26, 2015, Saturday, 23:55

In this project you will implement a very simple file system, called vsfs. It will be implemented as a library (static library: **libvsfs.a**). An application linked with that library will be able to use it to create and use files in a virtual disk. The virtual disk will be a file (of some certain size) created in your default Linux file system. That file will be acting as the storage, which will be managed by your vsfs file system. You will develop your project in Linux/C. The project can be done in groups of 2.

The vsfs file system will use FAT (file allocation table) method to keep track of the allocated and free blocks. It will use a simple directory organization: there will be one directory (i.e., one sub-directory), namely a root directory, and no other directory.

Your file system (library) will implement the following functions:

- **int vsfs_format(char *vdisk, int dsize).** Format the disk (virtual disk, i.e., a file in Linux acting as disk storage) named vdisk. The disk has size dsize. If success, returns 0, otherwise -1.
- **int vsfs_mount(char *vdisk).** Mount the filesystem. That means prepare it to be used by subsequent operations. Mounting may involve, for example, getting the FAT from disk into memory, initializing an open file table, and initializing some other in-memory structures and variables. Returns 0 if success; otherwise returns -1.
- **int vsfs_umount().** Unmount the file system.
- **int vsfs_create(char *filename).** Create a file of name filename. This involves creating/allocating a directory entry and an FCB (file control block). If you wish, you can combine FCB info into the directory entry for the file. That means you can put the attributes of the file into its directory entry. Then you will not need FCB.
- **int vsfs_open(char *filename).** Open file filename. Returns a non-negative integer file descriptor (file handle) if success. Otherwise returns -1. The file descriptor returned will be used by subsequent operations on the file. It can be simply the index of the open file table entry allocated for the file that is opened.
- **int vsfs_close(int fd).** Close the file whose descriptor (handle) is fd. If success returns 0, otherwise returns -1.
- **int vsfs_delete(char *filename).** Delete the file. If success returns 0, otherwise returns -1.
- **int vsfs_read(int fd, void *buf, int n).** Read (get) n bytes from file with handle fd. The read bytes (the bytes that are retrieved) are put into application buffer (array) buf. Space must have been allocated for buf by the application. Returns the number of bytes read (which can be less than n), if success, -1 otherwise. File position pointer is advanced by the number of bytes read.

- **int vsfs_write(int fd, void *buf, int n).** Write n bytes from application buffer buf into file fd. The file position pointer will be advanced by the number of bytes written. If success, returns the bytes read, otherwise returns -1.
- **int vsfs_truncate(int fd, int size).** Reduce the size of the file to size. Deallocate blocks if necessary. If size of the file was smaller or equal to the size parameter, the call has no effect. Returns 0 if success, -1 if error.
- **int vsfs_seek(int fd, int offset).** Set the file position pointer to offset. If offset is bigger than or equal to the file size, file position pointer is set to the file size.
- **int vsfs_filesize (int fd).** Returns the size of the file fd.
- **void vsfs_print_dir().** Print the files in the directory. One filename per line.
- **void vsfs_print_fat().** Print the names and block numbers of the blocks allocated to files. Format will be like the following example (assume there are two files file1 and file2):

  file1: 100 90 300
  file2: 3560 98 2400 567

Use the following constants:

```
#define BLOCKSIZE         4096     // bytes
#define MAXFILECOUNT      128      // files - max that FS can support
#define MAXDISKSIZE       (1<<28)  // 256 MB - max that FS can support
#define MINDISKSIZE       (1<<20)  // 1 MB - min that FS need
#define MAXFILENAMESIZE   128  // characters - max that FS can support
#define MAXBLOCKCOUNT     (MAXDISKSIZE / BLOCKSIZE)
#define MAXOPENFILES      128      // files
#define MAXREADWRITE      1024     // bytes; max read/write amount
```

Some initial source files will be provided in the following address so that you can start more easily.

- https://github.com/korpeoglu/p5

Submission:

Use Moodle.

Clarifications:

- You can use fixed size directory entries.
- You will access the disk in blocks. We are providing you two functions in the library that in github: getblock and putblock. You will use these access (read or write) a block in the disk (virtual disk). You will not access the disk other means. Just use these two functions.
- We provide you a program called makedisk that can be used to create a virtual disk of given size (i.e., a Linux file).
- We provide you a program called formatdisk that is calling vsfs_format library function to format the disk (formatting means installing/initializing the file system on disk). When you develop the vsfs_format function in the library, then this program will format a disk.

- There is an application provided (app.c). You can modify it. It is a test program. You can develop such applications to test your file system. We will also wrote our test applications to test and stress your library.