Operating Systems Laboratory
Spring Semester 2018-19

Assignment 6: Implement a Memory Resident File System

Assignment given on: **01/04/2011**

Assignment deadline: **08/04/2011, 2:30 PM (for Alternative 1)**

15/04/2011, 2:30 PM (for Alternative 2)

It is required to implement a memory-resident file system in a memory block. The disk will be simulated as a set of contiguous blocks in memory. The file system will have the following features:

- The file system has to be created on a dynamically allocated memory block (created using **malloc()** for example).
- The size of the file system and the block size will be taken as inputs during file system creation. Typical value can be: 64MB and 1KB respectively.
- The first block (Block-0) in the file system will contain the ***Super Block***, and will contain relevant information about the file system.
- The ***Super Block*** will contain information about the file system, like disk block size, total size of the file system, volume name, and other relevant information (mentioned below).

The following two alternatives have to be implemented:

Alternative 1: (Linked List Implementation using FAT)

1) The free blocks are maintained as a bit vector stored in the super block. The number of bits will be equal to the number of blocks in the file system. If the i-th bit is 0, then block i is free; else, it is occupied.
2) The data blocks of a file are maintained using a system-wide File Allocation Table (FAT), which will be stored in Block-1.
3) The directory is stored in a fixed block (with pointer in super block), and assume single-level directory. Each directory entry contains a number that is an index to FAT, and indicates the first block in the file. If the i-th entry of FAT contains j, this means block-j logically follows block-I in the file.
4) The data blocks will be stored from Block-2 onwards.

Alternative 2: (Indexed implementation using i-node)

1) The free blocks are maintained as a linked list of the blocks. The superblock will contain a pointer to the first free block. The last free block will contain -1 in the pointer field.
2) The data blocks of a file are maintained using index nodes or i-nodes. Each i-node will contain information about the data blocks, and will include 5 direct pointers, 1 singly indirect pointer, and 1 doubly indirect pointer. Each pointer will be 32 bits in size, and will indicate a block number. It will also store a ***type*** field indicating whether the file is a regular file or a directory, and ***file size*** in bytes. The i-nodes will be stored in Block-1 and Block-2, in increasing order of their numbers (i.e. i-node-0 first, followed by i-node-1, and so on).

3) The content of a directory file will be as follows. It will contain an array of records, each of size 16 bytes. The first 14 bytes srores the file name, and the last 2 bytes stores the inode number. Each directory will have two special entries “.” and “..”, indicating the current directory and the parent directory, respectively. For a block size of 512 bytes, 32 directory entries can be stored in each block.

The following API’s need to be supported for both the alternatives in the form of user-invocable functions from a C / C++ program. Define the parameters of the API functions appropriately.

- **my_open** open a file for reading/writing (create if not existing)
- **my_close** close an already open file
- **my_read** read data from an already open file
- **my_write** write data into an already open file
- **my_mkdir** create a new directory
- **my_chdir** change the working directory
- **my_rmdir** remove a directory along with all its contents
- **my_copy** copy a file between Linux file system and your file system
- **my_cat** display the contents of the specified file

Evaluation Guidelines:

While entering marks, the partwise break up should also be entered according to the marking guidelines given below. There is a separate component for individual assessment, based on how the student answers questions

Sl	Item	Marks
1	Representation of free blocks for FAT	5
2	APIs for FAT based allocation	36
3	Overall implementation for FAT allocation	4
4	Representation of free blocks for i-node	5
5	APIs for i-node based allocation	36
6	Overall implementation for I-node based allocation	4
	Total	90