Slower File System

Report

• Objective:

In this project, we will build a user-level library, libFS, that implements a good portion of a file system. Our file system will be built inside of a library that applications can link with to access files and directories.

• Things to do:

In the libFS library, we have three parts to implement, one is generic, one for files and one for directories. We have to make functions which can be used by other functions, to access our file system. We will discuss all three parts sequentially, as follows,

Generic File System API:

This part has only two functionalities, they are,

1. int FS_boot(char *path): This function will boot the whole file system, i.e, using the path provided as argument, it will get the file on which the image of file system is stored, and then load that file system, in case file does not exist, then it will create that file and store the file system image in it. Upon success return 0, on failiure return -1.

2. int FS_sync(): This function will just store the current image of file system, to the file pointed by FS_boot(), i.e, the

file which will be usedfor booting up. Upon sucess it will return 0 and on failiure return -1.

File access APIs:

This part will have functions regarding access of files, as:

1. **int File_create(char* file):** It will create the file with the filename provided by argument 'file'. If file with that name already exists then, return -1, else create file and return 0. File should not be open after createing the file, only file should be created.

2. **int File_open(char *file):** It will open the file, pointed by the argument 'file', and return the file desriptor. Which can be used, to access that file. If the file does not exists, then return -1.

3. **int File_Read(int fd, char *buffer, int size):** It will read the file,with file descriptor fd, data red should be in buffer, and number of bytes should be less than(in case of end of file) orequal to 'size'. Reading should begine from the current position of 'file pointer', and update it after reading is done. Before reading the file,it should be open. Number of bytes red should be returned,in case offileis not open return -1.

4. **int File_Write(int fd, char *buffer, int size):** It will write 'buffer' of size equal to 'size'(argument) in the file, with file descriptor 'fd'. Before writting fileshould be open, if it is not open then return -1. Writting should be done from the current position of 'file pointer'. Number of bytes written should be returned.

5. **int File_Seek(int fd, int offset):** It will seeks the file pointer, of the file with file descriptor ‘fd’, to ‘offset’ bytes ahead from current position. Again, file should be open before performing seek, if fileisnot open then return -1. If offset is greater than file size the angain return -1.

6. **int File_Close(int fd):** It will close the file, with file descriptor ‘fd’, if file is not currently open, then return -1. Upon success return 0.

7. **int File_Unlink(char *file):** It will delete the file pointed by argument ‘file’, and freed up all the data blocks used by this file.If fiedoes not exists then return -1. Upon success return 0.

Directory access APIs:

1. **int Dir_Create(char *path) :** It will create a directory, at the given path, by creating a file(of type directory), and making an entry to the parent directorie’s inode. In case of faliure return -1.

2. **int Dir_Size(char *path):** It ill return the size of the directory.

3. **int Dir_Read(char *path, char *buffer, int size) :** It would be used to read the directorypointed by argument ‘path’, it would return the directories entry in the argument ‘buffer’(including directories and files in the directory) of size of argument ‘size’. If entries are of more size than argument ‘size’, then return -1.

4. **int Dir_Unlink(char *path) :** Dir_Unlink() removes a directory refered by argument ‘path’, by freeing up its inode and data blocks, and removing its entry from the parent directory. Upon success, return 0. It would unlink only the empty directories, if there is any file in it then return -1.

• Assumptions :

- All the paths are absolute
- inode don’t have any indirect addresses
- Maximum size of file name is 16 bytes (15 bytes for name and 1 byte for ‘end-of-string’ delimiter, legal characters in file names are “.”, “-”, “_” but it does not support spacing
- Maximum length of path can be of 256 characters
- File_Write() isthe only waytoextend the file size
- Dir_Create() is not recursive, i.e., for a/b/c we have to create first a/then a/b/and then a/b/c/.
- inode have only three entries, size, type and data block pointers
- Data blocks are of same size of disk sector
- Each file can have maximum size of 30 data blocks
- File system will not perform any type of caching
- We have 10,000 blocks
- Upper limit of file/directory in the file system will be 1000
- Size ofa sector is 512B
- Size of blockis same as size of sector

- **Approach:**

  - The first block of our file system will be superblock, which will contain a magicnumber(randomly generated number), which will be key to the file system

  - Second block will be bitmap for inode blocks, which will tell us that which inode block is free, so that we can use them while creating file or directory.

  - Third block will be bitmap for data blocks, which willstoretheinfo about the availability of data bocks in the file system

  - Root directory's inode should be well known to the file system, hence it would also be stored in, fourth block

  - We have to maintain an open file table, which will contain the list of open files, there descriptors, inode number, and the file pointer, the table will have four columns, and file descriptor will be serial number on which that's file has entered in table.

  - Since, size of inode could be of maximum 62 bytes, as any address need 2 bytes, and hence 30 addresses need 60 bytes, and 2 bytes more,one for type and one for size, and size of block is 512B, and our file system can have 1000 files/directories, hence 1000 inodes, so 62 * 1000 = 62000 bytes are required for inodes, hence,

Number of blocks = 62000/512= 121

121 blocks after the bitmaps, will be reserved for inodes

Total blocks left for files/directories = 10000 – 121 – 3 = 9876

*Root inode's entery will be the first entry in the blocks reserved for inodes

- We have to maintain a global variable for 'error number', on different circumstances it should be set accrodingly, so that we can show the appropriate error(defined in provided API) to the user

• Timeline(from now) :

- In first week we will try toimplement the file access API, which will include, inode desinging, open file table implementation and implementation of different file accessfunctions

- In second week, we will move to directory access API, and testing of file access APIs.

- In third week, we will improve if any improvement isneeded in our file system, and then we will move to make the report of whole project