IT 308 OPERATING SYSTEMS
PROJECT REPORT

SINGLE LEVEL DIRECTORY FILE SYSTEM

<u>GROUP MEMBERS:</u>

ABHINAV GADDE - 201401017

ABHISHANTH PADARTHY - 201401093

INDEX

OVERVIEW:

The project involves creating a virtual disk and a simple file system on the top of the virtual disk. The implementation includes a library that offers a set of basic file system calls such as creating file, open file, closing file, write into the disk, read from the disk etc. The file data and file system meta-information are stored on the virtual disk. The virtual disk is a single file that is stored on the "real" file system provided by the Linux operating system. That is, you are basically implementing your file system on top of the Linux file system.

The project implementation has been done in **C** language. The code is written in ubuntu software.

DESCRIPTION:

The virtual disk has 8192 blocks. The size of each block is 4KB. Hence, the storage capacity of the entire file system is 32MB (5120 * 4 * 2^10).

The file system supports simple calls such as creating file, reading the data from the file, writing data in a file, size of the file etc.

Our implementation only includes a single directory and does not support hierarchical directories. All the files are stored in a single root directory on the virtual disk.

The allocation of the files in the memory is based on **First-Fit** concept i.e. the system searches for the required space of the file from the start of the location and if it finds any such block, it stores the file in that location.

We can also keep track of number of free blocks at every instant i.e. the number of blocks are free whenever a file is created or deleted.

We used **meta** structure to store the metadata of the file such as information like file name, file size, whether the file is opened or closed etc.

All the metadata will be stored in the block 0. The id of the next block is stored in the last 4 bytes of the current block and when the next block is not present, a value of **-1** is stored. Finally, we print all the blocks assigned to the file, size of the file and list the files on disk.

METHODS:

We have to provide the following three functions in order to manage the file system.

- **make_fs(char *disk_name) :** This function takes disk name as parameter and creates an empty file system on the virtual disk with name disk_name. The function returns **0** on success and **-1** when the disk_name could not be created/opened/initialized.
- **mount_fs(char *disk_name):** This function takes disk name as parameter and mounts a file system that is stored on the virtual disk with name disk_name. The function returns **0** on success and **-1** when the *disk_name* could not be opened or when the disk does not contain a valid file system (that you previously created with make_fs).
- **umount_fs(char *disk_name):** this function takes disk name as parameter and unmounts the file system from the virtual disk with name disk_name. The function returns **0** on success and **-1** when the disk disk_name could

not be closed or when data could not be written to the disk (this should not happen).

The following system functions are to be implemented for the file system to perform different operations.

- **fs_open(char *name):** This function takes file name as parameter and the file specified by name is opened for reading and writing and the file descriptor corresponding to this file is returned to the calling function. This function returns the **file descriptor** which is a non negative value on success and returns **-1** on failure.
- **fs_close(int fildes):** This function takes file descriptor as parameter and the file descriptor filedes is closed. This function returns **0** on success and returns **-1** if the file descriptor does not exist or it is open.
- **fs_create(char *name):** This function takes file name as parameter and creates a new file with name *name* in the root directory of your file system. This function returns **0** on success and returns **-1** on failure when the file with *name* already exists or when the file name is too long (exceeds 15 characters) or when there are already 64 files present in the root directory.
- **fs_delete(char *name):** This function takes file name as parameter and deletes the file with name *name* from the root directory of the file system and frees all data blocks and meta-data corresponding to that file. This function returns **0** on success and returns **-1** on failure when the file with *name* does not exist or when the file is currently open.
- **fs_read(int fildes, void *buf, size_t nbyte):** This function takes file descriptor, buffer and no. of bytes as parameters and read *nbyte* bytes of data from the file referenced by the descriptor filedes into the buffer pointed to by buf. This function returns **-1** on failure when file descriptor *fildes* is not valid.
- **fs_write(int fildes, void *buf, size_t nbyte):** This function takes file descriptor, buffer and no. of bytes as parameters and write *nbyte* bytes of data to the file referenced by the descriptor *fildes* from the buffer pointed

to by buf. This function returns the number of bytes that were written on success and returns **-1** on failure when the file descriptor *fildes* is not valid.

- **fs_get_filesize(int fildes):** This function takes file descriptor as parameter and returns the current size of the file pointed to by the file descriptor *fildes*. This function returns **-1** when the filedes is invalid.

OPTIMIZATION:

We have tried to improve the efficiency (in terms of time) of the system by implementing **First Fit** allocation of the files. The system searches for the sufficient space on the virtual file system to store the particular file from the starting location. When the system gets the sufficient space, it stores the file at that location. When it wants to store another file, the system searches again from the starting location and when it finds sufficient space, it stores the file there.

In best fit method, the system searches the entire memory even if it finds the memory blocks with sufficient space to store the file. So, when the files are large, it becomes a lot slower through this method to store the files.

APPLICATIONS:

- With the virtual file system, we can get an idea of number of free blocks are present at any instant and how many files can we store further.
- The file system can maintain a cache of directory lookups to enable easy location of frequently used directories.