Group no – 49
Prajwal Singhania – 15CS30043
Tanay Bhartia - 15CS30036

Design Report

Design of the memory –

In the basic design we have assumed a fixed number of inodes – 1000. Below is the list of all the important data structures created and their description:

Relevant Structures used

STRUCTUTRE	ATTRIBUTE DESCRIPTIONS	STRUCT DESCRIPTION

**typedef struct sup{
int totalsize;
int max_inodes;
int act_inodes;
int max_blocks;
int act_blocks;
std::bitset<MAX_INS> inode_map;
std::bitset<MAX_MEM> block_map;
}super_t;**
	->Total size of the mrfs
	->Maximum inodes(=1000)
	->Actual inodes in use
	->Max no. of data blocks
	->Actual #datablocks in use
	->Bitmap to keep track of used and free inodes
	->Bitmap to keep track of used and free datablocks
		Super Block structure

**typedef struct i{
char filename[MAX_FILENAME];
filetype file_t;
int filesize;
time_t last_modified;
time_t last_read;
mode_t mode;
int db_pointer[MAX_PTR];
}inode_t;**
	->Name of the file
	->Filetype: directory/regular
	->Size of the file(for dir = 0)
	->time of last modification
	->time of last access
	->access permissions
	->pointers – 8 direct, 1 indirect and 1 double indirect
		Single Inode Structure

**typedef struct list{
inode_t list[MAX_INS];
}inode_list_t;**
	->List of inodes	Inode Block structure

typedef struct tab
{   int inode_no;
int byteoffset;
int rw;
}entry_t;
	->inode of the file corresponding to the fd
	->byte offset
	->read or write mode indicator
		Table entry for a file descriptor

typedef struct fs
{
char* mem;
super_t* super;
inode_list_t* inode;
data_block_t* db_blocks;
std::vector<entry_t*> table;
}mrfs;
	->pointer to the memory of the fs
	->pointer to the super block
	->pointer to the inode block
	->pointer to the series of datablocks
	->Active file descriptors table
		Struct for the mrfs

Functions –

API Functions:

FUNCTION	DESCRIPTION
int create_myfs(int size);	*Allocates memory for the filesystem *Creates and initialize the super block *Creates and initalize the inode_list block *Creates the root directory
int copy_pc2myfs(char* source, char *dest);	*Create the inode for the file *read data from the source file block wise and store in myfs following the structure of pointers - direct, indirect and double indirect *Add the file details to the current working directory
int copy_myfs2pc(char* source, char *dest);	*This fucntion copies the source file in current working directory of myfs the dest in pc *It first list outs the file in the current working directory and finds the matching file *If the file is found, it is copy to the destination otherwise -1 is returned
int rm_myfs(char* filename);	*Removes the specified file from the current working directory if present

int showfile_myfs(char* filename);	*Finds the file in the current working directory *Displays the file character by character *This function is only meant for text files
int ls_myfs();	*Prints the files in the current working directory
int mkdir_myfs(char* dirname);	*Creates a new directory in the current working directory *It first checks if the dirname already exists in the cwd and returns -1 if so *It then creates the new directory, adds the . and .. entries to it and add it to the current working directory
int chdir_myfs(char *dirname);	*Finds the given directory in the current working directory and changes to it as the cwd if valid
int rmdir_myfs(char* dirname);	*Recursivley removes a directory and all files and directories within it too
int open_myfs(char* filename, char mode);	*Opens the file in the specified mode - read or write *If the mode is read then it searches for the file in the current working directory. *If found it assigns a file descriptor to it and then stores the file descriptor along with relevant info in a table *If the mode is write then it removes the file(if present) with the corresponding name in the current working directory and creates a fresh file with the name *Again it assigns a file descriptor to it and then stores the file descriptor along with relevant info in a table
int close_myfs(int fd);	*Closes the file descriptor: *Checks if the file descriptor is valid or not *If it is valid, remove it from the table of active file descriptors
int read_myfs(int fd, int nbytes, char* buff);	

	*Reads from the given file descriptor(if valid) nbytes number of consecutive sites starting at the byteoffset stored with the fd *It stores the bytes in the buffer *In case the file ends before nbytes, it only reads upto the end of the file and returns *It returns the actual number of bytes read. In case of error, -1 is returned
int write_myfs(int fd, int n_bytes, char* buff);	*Writes nbytes number of bytes present in buf to the file correspoding to the file descriptor fd(if valid) starting at the byteoffset stored with fd *It returns the number of bytes actually written. In case of error, -1 is returned
int eof_myfs(int fd);	*Checks if the file descriptor has reached the end of file or not *The checking is done by comparing the byteoffset with the filesize *In eof is reached 1 is returned, 0 if eof is not reached, and -1 for error
int dump_myfs(char* dumpfile);	*Saves the whole filesystem to a file in pc
int restore_myfs(char* dumpfile);	*Restores the filesystem from a file in pc
int status_myfs();	*Prints the status of the file system
int chmod_myfs(char* name, int mode);	* Changes the access permisisions of the file/directory in current working directory if present. Otherwise returns -1

Helper Functions:

FUNCTION	DESCRIPTION
int next_inode();	*This function returns the first free inode *If no free inode is available it returns -1
int next_free_block();	*This function returns the first data block

	*If no free data block is available it returns -1
void ls(vector<ls_list_t>* list)	*Returns a list of the files in a directory along with the corresponing pointer to the particular file entry in myfs
int* search_directory_space(int dir)	*This function searches for a free space to the add the file information in the given directory - dir *It returns a pointer to the start if that free location
void clear_data(int inode_no, vector<int> allocated_b)	*Helper function to clean the data in case copy fails
void print_permissions(mode_t mode)	*Helper function to print the permissions in ls -l format
int get_pointer_by_index(inode_t node, int index)	*Helper function to locate data blocks for a file
int getnextfd()	*Helper function to get the next available file descriptor
int get_pointer_by_index_write(inode_t* node, int index)	*Helper function to get the data block pointer for the write function. It assigns a new datablock if necessary