CSE 512 OPERATING SYSTEMS

HW-3 Part-1

File System Design Report

In this homework we would design a filesystem that uses i-node structure. I tried to understand all about UNIX System 7 file system by reading the textbook well and I did some research to design a file system structure.

Segmentation

Super Block	i-nodes	I-node bitmap	Data Bitmap	Data Blocks

The segmentation of all filesystem is like upside. SuperBlock is at first block. We hold addresses of all segments in superblock to access later.

Super Block Struct

```c
typedef struct SuperBlock{            // SuperBlock Struct

    uint32_t    magic_number;                 /* File system magic number */
    uint8_t     block_size;                   /* Size of a block in KB */
    uint16_t    inode_size;                   /* Size of a i-node in bytes */
    uint16_t    num_of_i_nodes;               /* Number of i_nodes */
    uint32_t    num_of_disk_blocks;           /* Number of disk blocks */

    // segment adresses
    uint16_t    inode_start_addr;
    uint16_t    data_bitmap_addr;            /* the address of data block bitmap */
    uint16_t    i_node_bitmap_addr;          /* the address of i_node bitmap */
    uint16_t    data_block_addr;
    uint16_t    root_dir_index;
    char file_name[FILE_NAME_MAX];

}SuperBlock;
```

Superblock has magic number to detect if the given partition or disk is structured in own file system. Block size is used to index all data blocks. All segment addresses are holded here. To lookup data block addres or an i-node we need to get the starting segment point from super block then by referencing it we find the required block.

I-node Struct

Every I-node holds it's own data block addresses in the struct. To access the data we look-up those addresses. I also I-node type (is_directory) to determine if an I-node Is directory I-node or file I-node. I specified the I-node number as 2000. Because our disk is 16 MB at most as defined in the homework.

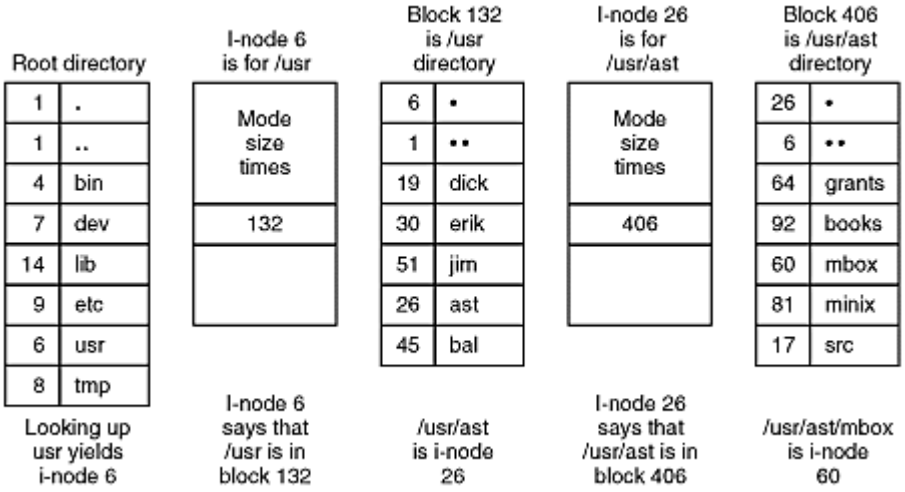
```c
typedef struct Inode
{
    short valid;
    size_t file_size_bytes;                     /* File size in bytes */
    struct tm  creation_time;                   /* Creation time in tm struct */
    struct tm last_access_time;
    struct tm last_modification_time;
    short is_direcory;                          /* If this i-node is directory or file  */
    char name[FILE_NAME_MAX];                   /* name of i-node */
    int index;                                  /* index number of i-node */

    uint16_t direct_blocks[DIRECT_BLOCK_SIZE];  /* Data block addresses */


}Inode;
```

Directory Structure

As you already know, in unix world directories are also a file. So those are also hold in I-node form. But we need to hold a hierarchical structure for directories. For this purpose, I designed that, we hold the root directory I-node index in superblock and every directory I-node holds entries of sub directories and files in it's own data blocks in a struct of DirectoryEntry as mentioned in the textbook.



Directory Entry

I-Node Num.(2 byte) (index)	File Name (14 byte)

All directories have some directory entries. As default all of them have "." and ".." entries.

Directory Lookup

For finding a path in directory structure is like that, firstly starts with root directory then it finds sub directory I-node number then goes to the I-node and then find sub directory and so on. I implemented this with using recursion.

Free Blocks

I used a bitmap structure for free blocks. If we have 4000 data blocks then we have a bitmap like this:

4000 blocks represented as 4000 bits in bitmap. Every

4000 bits = 4000 / 8 = 500 bytes

We have 500 bytes for 4000 data blocks. This is variable according to number of blocks.

As I showed at segmentation part. I kept 2 bitmaps in my file system. One is for data blocks and the other one is for I-nodes. They are placed before data blocks and after I-nodes.

File System Operations

Disk operations

I abstracted the disk in a disk-structure that holds a file pointer, block size, super block and is_open for mounting and also base disk functions like create, open, close. These operations is realized in a file as mentioned in hw pdf.

```
int read_data_block(Disk *disk, int block_index, void* buffer, int size);
int write_data_block(Disk *disk, int block_index, void *data, size_t size);
int reserve_block(Disk* disk, short reserve_type);

uint8_t disk_read_byte(Disk *disk, int address);
void disk_write_byte(Disk *disk, int address, uint8_t byte);

void set_bit(Disk* disk,int isData, unsigned int index, unsigned int bit);
uint8_t get_bit(Disk* disk, int isData, unsigned int index);


Inode* read_inode(Disk* disk, int inode_index);
void write_inode(Disk* disk, Inode* inode);
int find_INode(Disk* disk, char* path, int start_inode);

void write_inode_data(Disk* disk, Inode* inode, void* data, size_t size, int block);
int read_inode_data(Disk* disk, int inode, void** data, int size);
```

Base block Operations

I implemented some functions like read-write at spesific index of data blocks, I-node or bitmaps as shown downside. Using these functions I implemented all file system operations like mkdir, dir etc.

```
int read_data_block(Disk *disk, int block_index, void* buffer, int size);
int write_data_block(Disk *disk, int block_index, void *data, size_t size);
int reserve_block(Disk* disk, short reserve_type);

uint8_t disk_read_byte(Disk *disk, int address);
void disk_write_byte(Disk *disk, int address, uint8_t byte);

void set_bit(Disk* disk,int isData, unsigned int index, unsigned int bit);
uint8_t get_bit(Disk* disk, int isData, unsigned int index);


Inode* read_inode(Disk* disk, int inode_index);
void write_inode(Disk* disk, Inode* inode);
int find_INode(Disk* disk, char* path, int start_inode);

void write_inode_data(Disk* disk, Inode* inode, void* data, size_t size, int block);
int read_inode_data(Disk* disk, int inode, void** data, int size);
```

Normal File System Operations

```
int add_file(Disk* disk, int dest_inode_num, Inode* new_inode);          // adds the given i-node to the given destination directory
void dir(Disk* disk, char * path);                                       // lists all files and directories in the given path
void mkdir(Disk* disk, char* path);                                      // makes a directory in the given path
Inode new_file(Disk* disk, char name[FILE_NAME_MAX], int isDirecory);    // creates a inode in the disk and returns the inode
int write(Disk* disk, char* path, char* file_name);                      // creates a file and writes the contents of the other file
int read(Disk* disk, char* path, char* file_name);                       // reads the given file and writes it to linux file
int rm_inode(Disk* disk, char* path, int isDirectory);                   // removes the given i-node by specifying the inode is directory or not.
int del(Disk* disk, char* path);                                         // deletes the file at specified path
int rmdir(Disk* disk, char* path);                                       // reamoves the directory at specified directory.
```

int add_file (Disk* disk, int dest_inode_num, Inode* new_inode)

- Adds the directory entry to the destination I-node

Inode new_file(Disk* disk, char name[FILE_NAME_MAX], int isDirecory)

- Reserves a I-node block with reserve_block()
- Creates and initializes the I-node
- Writes the I-node to the disk

void dir(Disk* disk, char * path)

- Finds the given I-node of specified path with find_Inode()
- Reads all directory entries with Read_inode_data()
- Prints every directory entries with required features.

void mkdir(Disk* disk, char* path)

- Splits the path and file name with split_path() function
- Finds the destination I-node number of directory
- Creates new directory with new_file(is_directory = 1)
- Adds the file I-node to the destination directory

int write(Disk* disk, char* path, char* file_name)

- Splits the path to direcory path and file name with split_path().
- Finds the destination I-node of directory.
- Creates new file with new_file().
- Adds the file to the destination I-node of directory.
- Reads the given linux file_name

- Writes the data to the file with write_inode_data().

int read(Disk* disk, char* path, char* file_name)

- Finds the given I-node of given file path with find_Inode().
- Reads the data of the I-node read_inode_data().
- Writes to the data to the given linux file with fprintf().

int rm_inode(Disk* disk, char* path, int isDirectory)

- Splits the path to the directory and file_name with split_path().
- Deletes the directory entry of file of directory if directory is empty.

int del(Disk* disk, char* path)

- deletes the file by using rm_inode(isDirectory = 0).

int rmdir(Disk* disk, char* path)

- Removes the directory by using rm_inode(isDirectory = 1).
- This is works If the directory is empty like ms-dos commad line.

Conclusion

In this homework we made a filesystem. I tried to do everything best. I thought the hole structure 2-3 days. I hope and sure you will like the implementation structure.