CS518:OPERATING SYSTEMS

[A2 Write Once – Project Report]

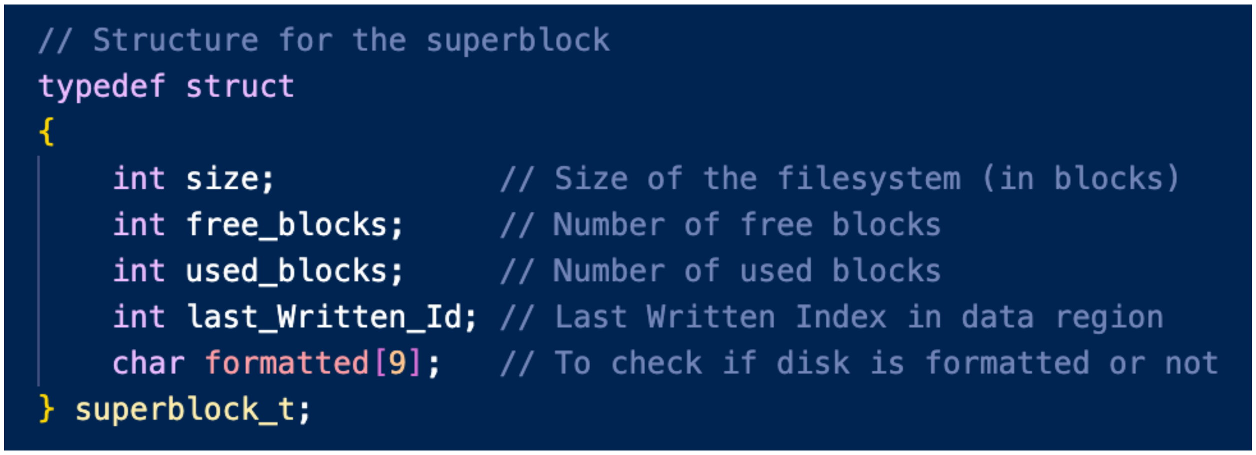
Anubhav Garikapadu **[NetID: AG2112]**
Vijay Swaminathan **[NetID: VS797]**

Abstract:

Implementing a write-once filesystem that exists entirely inside one 4MB file in the native operating system. On mount, the library opens the file that holds the 'disk' and reads all 4MB of it into memory. For the lifetime of the Process that mounts the filesystem, all reads and writes are served using the image in memory. When the filesystem is unmounted, the library will write out the entire 4MB image of modified 'disk' to same file it came from, overwriting the old information and preserving any changes made.

A. Data Structures:

1. Superblock structure

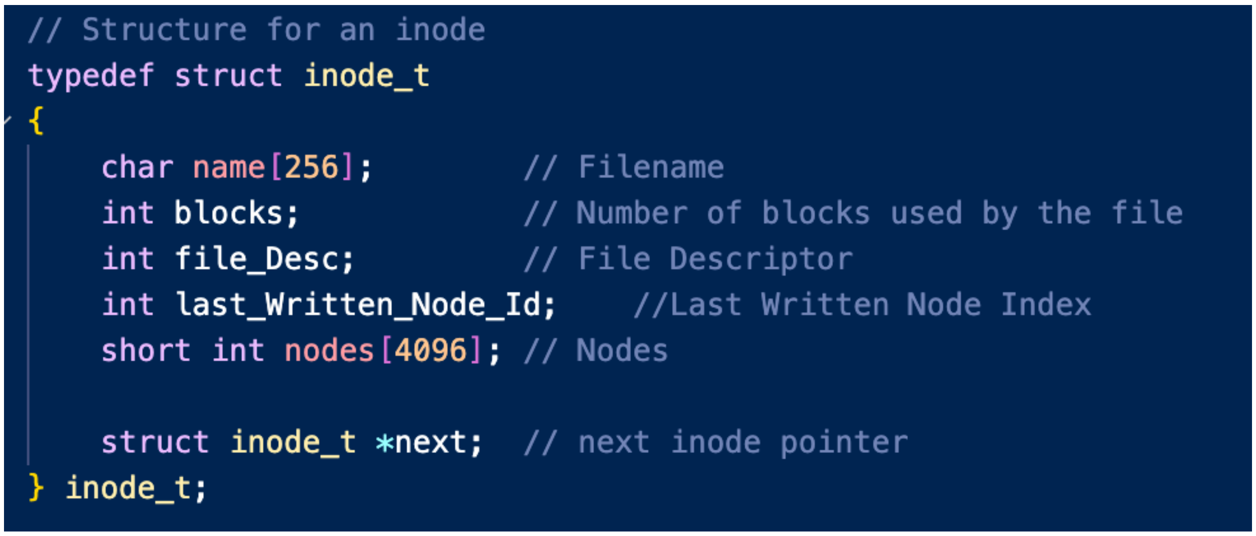


```c
// Structure for the superblock
typedef struct
{
    int size;            // Size of the filesystem (in blocks)
    int free_blocks;     // Number of free blocks
    int used_blocks;     // Number of used blocks
    int last_Written_Id; // Last Written Index in data region
    char formatted[9];   // To check if disk is formatted or not
} superblock_t;
```

The superblock data structure consists of 4 variables and a character array:

- The size variable represents the size of the filesystem in blocks
- free_blocks represents the number of blocks that are unutilized.
- used_blocks likewise represents the number of utilized blocks.
- The last_Written_Id variable represents the last written index in data region.
- The formatted string checks whether the disk is formatted/not formatted.

2. Inode structure



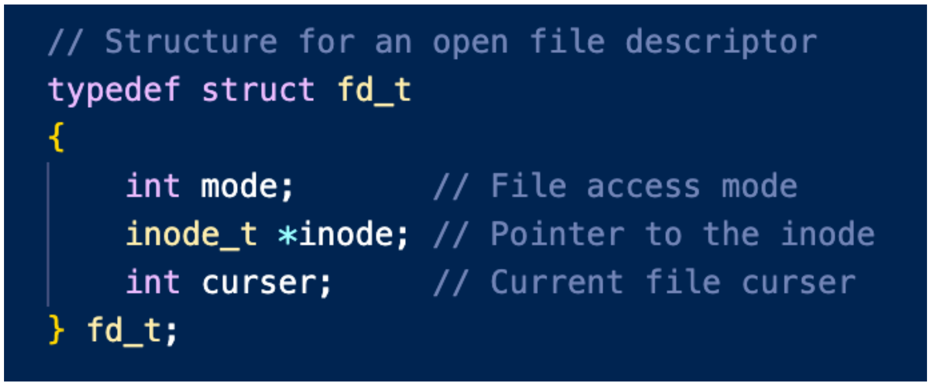
```c
// Structure for an inode
typedef struct inode_t
{
    char name[256];          // Filename
    int blocks;              // Number of blocks used by the file
    int file_Desc;           // File Descriptor
    int last_Written_Node_Id;    //Last Written Node Index
    short int nodes[4096]; // Nodes

    struct inode_t *next;  // next inode pointer
} inode_t;
```

The inode linked list data structure consists of:

- Int blocks which indicates the number of blocks used by the file.
- Int file_Desc indicates the file descriptor
- Int last_Written_Node_Id represents the last written node index.
- The name string represents the filename
- Nodes array represents the data blocks in the data region.

3. File Descriptor structure



```c
// Structure for an open file descriptor
typedef struct fd_t
{
    int mode;          // File access mode
    inode_t *inode;    // Pointer to the inode
    int curser;        // Current file curser
} fd_t;
```

The file descriptor data structure consists of:

- Mode variable that indicates the different file access modes namely:
  - WO_RDONLY (read only- 1)
  - WO_WRONLY (write only- 2)
  - WO_RDWR (read and write - 3)
- *inode is a pointer to the inode.
- Curser represents the current data block.

4. Additional Data Variables

- File Descriptor Table: fd_table[max_files] represents the file descriptor table for open files with maximum size set to 100
- char *data_region respresnts a pointer to the start of the data region in memory.
- inode_t *inode_region provides a pointer to the start of the inode in memory.
- superblock_t *superblock_region provides a pointer to the start of the superblock region in memory
- max_fs_size – represents maximum size of the file system in bytes.
- Block_size is 1024 bytes.
- char *filesystem
- char *disk_Name

B. Functions:

1. Int wo_mount (char *diskname, void *memory_address)

   - Mounts the file system from the disk.
   - Opens the file for reading and returns error no (file access error) if fp is NULL.
   - Read the entire file system into memory
   - Set up pointers to the different memory regions

2. Int wo_unmount (void *memory_address)

    - Open the disk for writing and returns error no (file access error) if fp is NULL.
    - Writes the entire filesystem from memory to the disk.

3. Int wo_open(char *filename, int flags)

    - Opens the file and finds the first empty slot in the file descriptor table.

4. Int wo_create(char *filename, int flags)

    - When creating a file, checks if file is already present in the system.
    - If so, returns error no.
    - Opens the file and finds the first empty slot in the file descriptor table, similar to wo_open() function.

5. Int wo_read (int fd, void *buffer, int bytes)

    - Checks if the file descriptor is valid, else returns error.
    - Checks if the file is open in read mode, else returns error. This is done by checking if it is in write only (WRONLY) mode as the other two modes have read access.
    - Calculates the number of blocks to read
    - Checks if the file has enough data
    - Read the data

6. Int wo_write (int fd, void *buffer, int bytes)

    - Checks if the file descriptor is valid, else returns error.
    - Checks if the file is open in write mode, else returns error.
    - Calculates the number of blocks to write.
    - Checks if there is enough free space to write the data, if not, throws error.
    - Finds first available block and writes the data.
    - Updates the file size and the block count.

7. Int wo_close (int fd)

    - Checks if the file descriptor is valid
    - Throws error if it is invalid
    - Closes the file.