UNIX SYSTEMS PROGRAMMING LAB
FILE SYSTEMS PROJECT

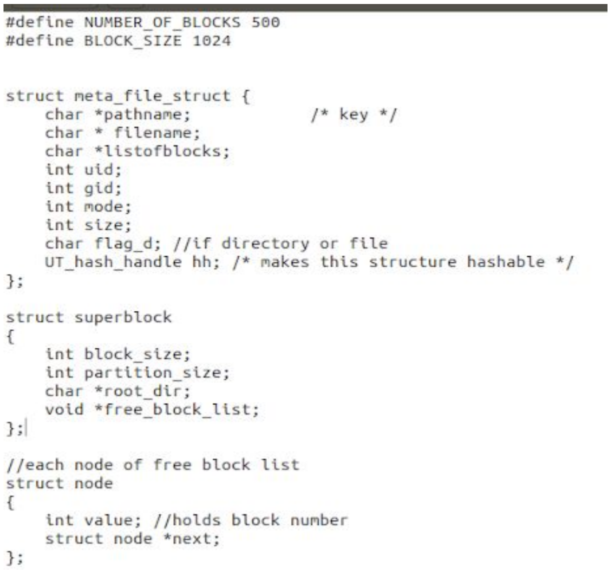
Niharika Misra	**01FB15ECS185**
Nikita Metha	**01FB15ECS191**
Niriksha Kunder	**01FB15ECS192**

AIM

To build a **simple file system on Unix, with or without FUSE.** Implement basic functionality, such as read/write on files, directory functions. File system should be **persistent**, in that, data should remain consistent even on remount.

PHASE 1

Implemented a simple file system using in memory emulator that we created.



```
#define NUMBER_OF_BLOCKS 500
#define BLOCK_SIZE 1024


struct meta_file_struct {
    char *pathname;            /* key */
    char * filename;
    char *listofblocks;
    int uid;
    int gid;
    int mode;
    int size;
    char flag_d; //if directory or file
    UT_hash_handle hh; /* makes this structure hashable */
};

struct superblock
{
    int block_size;
    int partition_size;
    char *root_dir;
    void *free_block_list;
};

//each node of free block list
struct node
{
    int value; //holds block number
    struct node *next;
};
```

Figure 1

The **data structures** used were

- **Linear array for mimicking memory**- each element of the array holds a **char*** of size **1024** bytes **(Block size that we used)** **--(1)**
- **Inode table to hold metadata** of files and directories was simulated using a **hash table of structs**. **--(2)**

- **Superblock** to hold root directory information, disk size, block size and a pointer to free block list. (**structure malloc'ed on heap**). **--(3)**
- **Free block list** implemented as a linked list **(queue)** **--(4)**

Functions that we implemented:

- **void mknikfs():**

Called before FUSE handle in main which **handles the initialising of all data structures** and memory, on mount. (Creation of file system, **our implementation of mkfs()**)

- **void initialize_mem_disk():**

Called by **mknikfs()** to create **(1)**

- **int write_block(char *write_data, int block_num):**

Given data and block number, writes data to that block. Returns 1 on success, else 0.

- **char* read_block(int block_num):**

Reads from data block. Returns NULL on error.

- **void clear_all():**

Free all malloc'ed memory on unmount.

- **int create_file(char *filename, char* text, char* dir):**

Simply just allocates memory for file, and returns file descriptor.

- **char* read_file(char *filename, int offset):**

Reads file at particular offset.

- **void initialize_superblock(char *root):**

Called by **mknikfs()** to create **(3)**

- **void write_file_meta(struct meta_file_struct *, char*, int,int,int):**

Create new metadata structure for new file, called on file creation.

- **void initialize_freeblocklist():**

Called by **mknikfs()** to create **(4)**

- **struct meta_file_struct* read_meta(char*):**

To read metadata when get_attr called by FUSE.

- **void create_dir(char* dname):**

To create new directory entry in memory and meta.

- **void app_dir(char* dname, char* filename):**

Called when a new file is created in a directory.

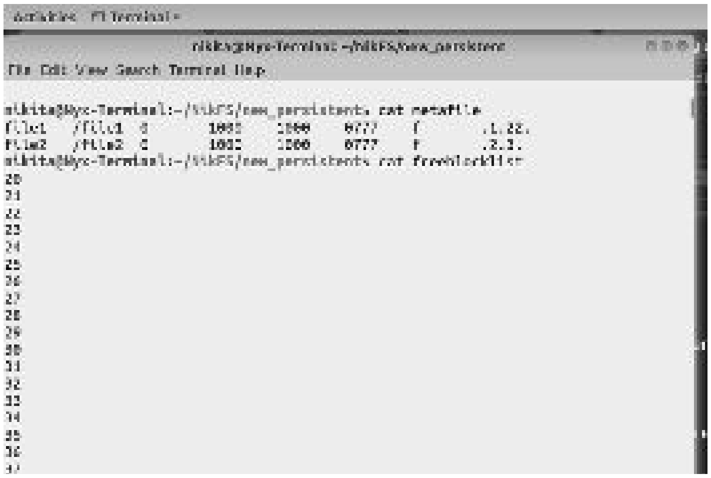
- **char* read_dir(char* name):**

Used when FUSE calls read_dir()

PHASE 2

For persistent store, we use **files** (instead of malloc'ed memory from heap), for all data structures created in Phase 1.
Files are:

- **Fusedata**: holds all blocks, like in (1)
- **Metafile**: to hold all metadata, just like in (2) where each line is entry of one file, each attribute separated by commas
- **Freeblocklist**: each free block number is stored in one line. Enqueue is just appending to file, while dequeue is truncating first entry of file.
- **Superblock_info**: to store superblock information, like in (3)


Activities   Terminal ▾
nikita@Nyx-Terminal: ~/NikFS/new_persistent
File Edit View Search Terminal Help

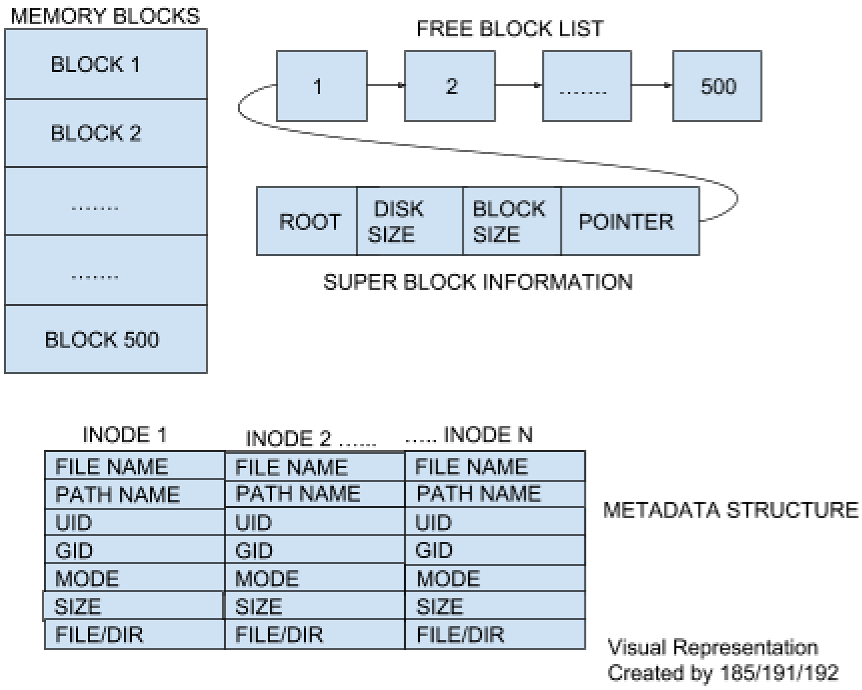
```
nikita@Nyx-Terminal:~/NikFS/new_persistent$ cat metafile
file1   /file1  0       1000    1000    0777    f       .1.22.
file2   /file2  0       1000    1000    0777    f       .2.3.
nikita@Nyx-Terminal:~/NikFS/new_persistent$ cat freeblocklist
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
```

Figure 2

PHASE 3

Implemented **basic calls required for test cases as given** (mkdir, readdir, open, close, read, write, copy of file).

VISUAL REPRESENTATION OF OUR DATA STRUCTURE



INODE 1	INODE 2	..... INODE N
FILE NAME	FILE NAME	FILE NAME
PATH NAME	PATH NAME	PATH NAME
UID	UID	UID
GID	GID	GID
MODE	MODE	MODE
SIZE	SIZE	SIZE
FILE/DIR	FILE/DIR	FILE/DIR

Figure 3

IMPLEMENTATION DETAILS

How have we stored blocks for each file?

As can be seen in **figure 2,** the list of blocks for each file is separated by '.'
This is used in readfile/writefile implementations. As new blocks are required on appending a file, all we need to do is access the metadata, and write a new free block number to this list (stored as a string).

What does the meta file look like?

Meta is stored as
<filename>\t **<pathname>** \t **<size>** \t **<uid>** \t **<gid>** \t **<mode>** \t **<flag>** \t
<blocklist>

How does hashing work with persistent store?

The metadata is **read in line by line** and stored in the same hash table (in memory as in phase 1) every time the system is mounted (**mknikfs()** is called). **Hash values can be different on every mount**, but it does not matter to us, as we only require pathname to be indexed.

LIMITATION OF USING HASH TABLE

Case of collisions has not been accounted for, thus there may be two or more files with same hash value, causing errors.

How is directory implemented?

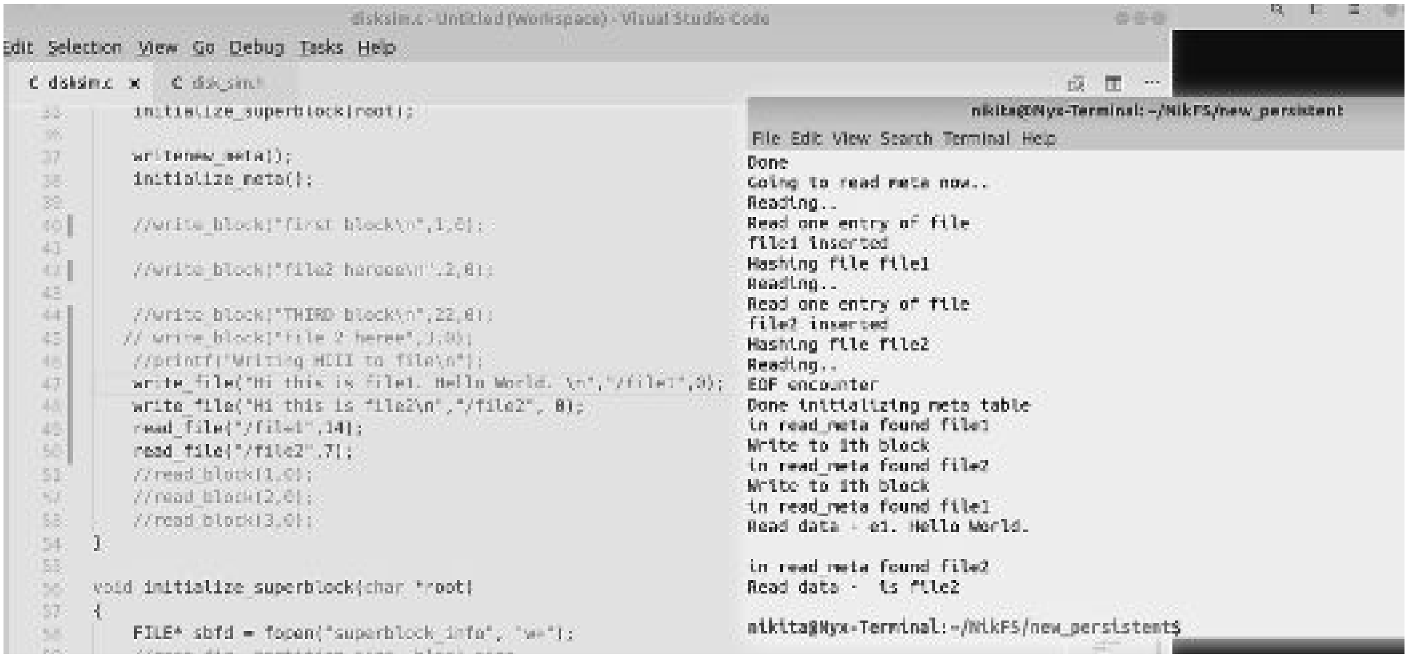
Directories' meta is on the same inode table as files. They are **identified by flag character as 'd'**. Directory blocks store all **filenames of the files in them, line after line**.

Features of our File System:

- Each **inode is stored as a hash table entry**, using the uthash implementation. Hashing is done based on **pathname as the key, for O(1) retrieval.**
- **Free block list stored as a queue,** such that **enqueue** (adding a free block on deleting a file) and **dequeue** (giving a free block to file for write) work as **O(1) operations.**
- A **simple character flag differentiates a directory meta from file meta**. Directories entries are the filenames that they contain.
- **There is no limitation to the maximum size of the file** (only limited by partition size minus the size of the directory it is in and other files.)

Experimentation:

- For using lseek in FUSE, we tried changing readfile and writefile functions to accept “offset” as an argument, which will print from anywhere between the file, and is able to create holes in the file.
  We weren’t able to link it to FUSE to use with our filesystem due to lack of time.
  Here is a screenshot of what we’ve accomplished with this, currently.


disksim.c - Untitled (Workspace) - Visual Studio Code
Edit Selection View Go Debug Tasks Help
C disksim.c ×
C disk_sim.h
nikita@Nyx-Terminal: ~/NikFS/new_persistent
File Edit View Search Terminal Help

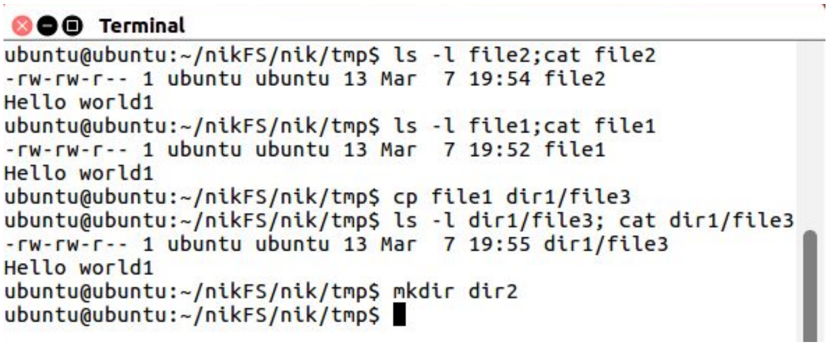
```
    initialize_superblock(root);

    writenew_meta();
    initialize_meta();

    //write_block("first block\n",1,0);

    //write_block("file2 hereee\n",2,0);

    //write_block("THIRD block\n",22,0);
   // write_block("file 2 heree",1,0);
    //printf("Writing HIII to file\n");
    write_file("Hi this is file1. Hello World. \n","/file1",0);
    write_file("Hi this is file2\n","/file2", 0);
    read_file("/file1",14);
    read_file("/file2",7);
    //read_block(1,0);
    //read_block(2,0);
    //read_block(3,0);
}

void initialize_superblock(char *root)
{
    FILE* sbfd = fopen("superblock_info", "w+");
```

```
Done
Going to read meta now..
Reading..
Read one entry of file
file1 inserted
Hashing file file1
Reading..
Read one entry of file
file2 inserted
Hashing file file2
Reading..
EOF encounter
Done initializing meta table
in read_meta found file1
Write to 1th block
in read_meta found file2
Write to 1th block
in read_meta found file1
Read data - e1. Hello World.

in read_meta found file2
Read data -  is file2

nikita@Nyx-Terminal:~/NikFS/new_persistent$
```

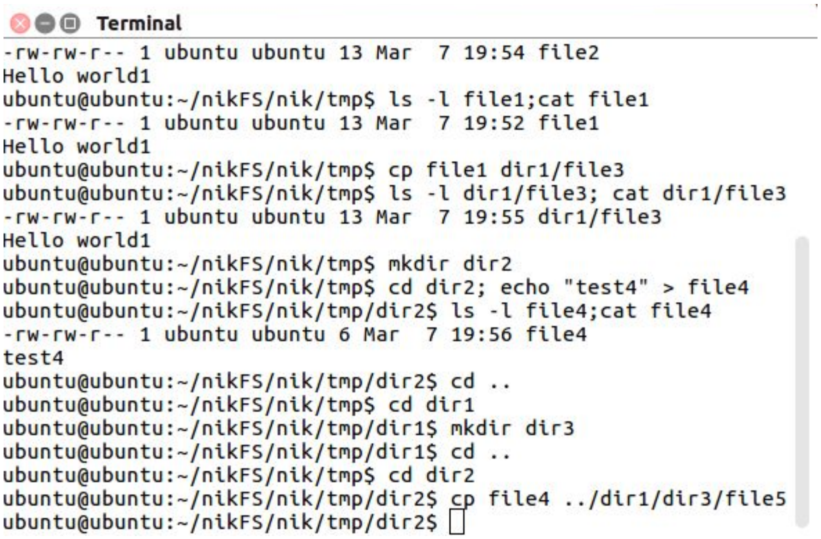
Future enhancements:

- We aim to implement functionality for other system calls that we have learned through the course of USP, mainly hardlinks and softlinks.

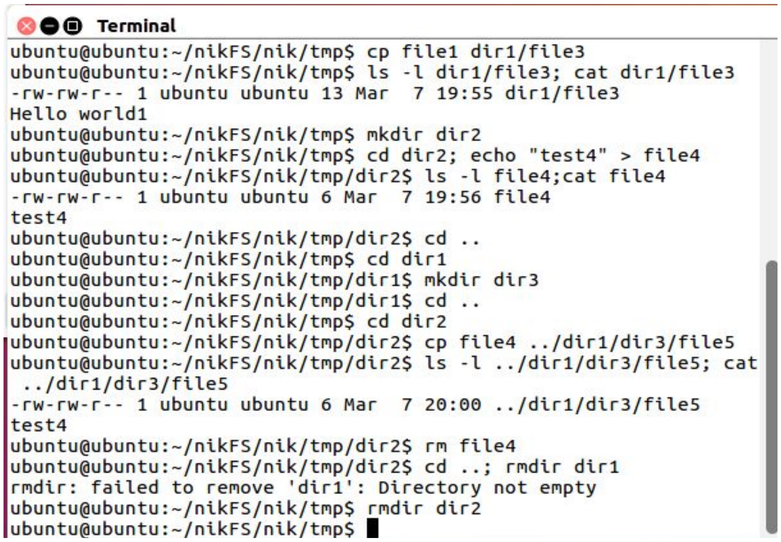
Screenshots of working File System



```
ubuntu@ubuntu:~/nikFS/nik/tmp$ ls -l file2;cat file2
-rw-rw-r-- 1 ubuntu ubuntu 13 Mar  7 19:54 file2
Hello world1
ubuntu@ubuntu:~/nikFS/nik/tmp$ ls -l file1;cat file1
-rw-rw-r-- 1 ubuntu ubuntu 13 Mar  7 19:52 file1
Hello world1
ubuntu@ubuntu:~/nikFS/nik/tmp$ cp file1 dir1/file3
ubuntu@ubuntu:~/nikFS/nik/tmp$ ls -l dir1/file3; cat dir1/file3
-rw-rw-r-- 1 ubuntu ubuntu 13 Mar  7 19:55 dir1/file3
Hello world1
ubuntu@ubuntu:~/nikFS/nik/tmp$ mkdir dir2
ubuntu@ubuntu:~/nikFS/nik/tmp$ 
```



```
-rw-rw-r-- 1 ubuntu ubuntu 13 Mar  7 19:54 file2
Hello world1
ubuntu@ubuntu:~/nikFS/nik/tmp$ ls -l file1;cat file1
-rw-rw-r-- 1 ubuntu ubuntu 13 Mar  7 19:52 file1
Hello world1
ubuntu@ubuntu:~/nikFS/nik/tmp$ cp file1 dir1/file3
ubuntu@ubuntu:~/nikFS/nik/tmp$ ls -l dir1/file3; cat dir1/file3
-rw-rw-r-- 1 ubuntu ubuntu 13 Mar  7 19:55 dir1/file3
Hello world1
ubuntu@ubuntu:~/nikFS/nik/tmp$ mkdir dir2
ubuntu@ubuntu:~/nikFS/nik/tmp$ cd dir2; echo "test4" > file4
ubuntu@ubuntu:~/nikFS/nik/tmp/dir2$ ls -l file4;cat file4
-rw-rw-r-- 1 ubuntu ubuntu 6 Mar  7 19:56 file4
test4
ubuntu@ubuntu:~/nikFS/nik/tmp/dir2$ cd ..
ubuntu@ubuntu:~/nikFS/nik/tmp$ cd dir1
ubuntu@ubuntu:~/nikFS/nik/tmp/dir1$ mkdir dir3
ubuntu@ubuntu:~/nikFS/nik/tmp/dir1$ cd ..
ubuntu@ubuntu:~/nikFS/nik/tmp$ cd dir2
ubuntu@ubuntu:~/nikFS/nik/tmp/dir2$ cp file4 ../dir1/dir3/file5
ubuntu@ubuntu:~/nikFS/nik/tmp/dir2$ 
```



```
ubuntu@ubuntu:~/nikFS/nik/tmp$ cp file1 dir1/file3
ubuntu@ubuntu:~/nikFS/nik/tmp$ ls -l dir1/file3; cat dir1/file3
-rw-rw-r-- 1 ubuntu ubuntu 13 Mar  7 19:55 dir1/file3
Hello world1
ubuntu@ubuntu:~/nikFS/nik/tmp$ mkdir dir2
ubuntu@ubuntu:~/nikFS/nik/tmp$ cd dir2; echo "test4" > file4
ubuntu@ubuntu:~/nikFS/nik/tmp/dir2$ ls -l file4;cat file4
-rw-rw-r-- 1 ubuntu ubuntu 6 Mar  7 19:56 file4
test4
ubuntu@ubuntu:~/nikFS/nik/tmp/dir2$ cd ..
ubuntu@ubuntu:~/nikFS/nik/tmp$ cd dir1
ubuntu@ubuntu:~/nikFS/nik/tmp/dir1$ mkdir dir3
ubuntu@ubuntu:~/nikFS/nik/tmp/dir1$ cd ..
ubuntu@ubuntu:~/nikFS/nik/tmp$ cd dir2
ubuntu@ubuntu:~/nikFS/nik/tmp/dir2$ cp file4 ../dir1/dir3/file5
ubuntu@ubuntu:~/nikFS/nik/tmp/dir2$ ls -l ../dir1/dir3/file5; cat
 ../dir1/dir3/file5
-rw-rw-r-- 1 ubuntu ubuntu 6 Mar  7 20:00 ../dir1/dir3/file5
test4
ubuntu@ubuntu:~/nikFS/nik/tmp/dir2$ rm file4
ubuntu@ubuntu:~/nikFS/nik/tmp/dir2$ cd ..; rmdir dir1
rmdir: failed to remove 'dir1': Directory not empty
ubuntu@ubuntu:~/nikFS/nik/tmp$ rmdir dir2
ubuntu@ubuntu:~/nikFS/nik/tmp$ 
```