CS 314 – Operating Systems Lab
Lab–10 Report

Student 1: Kavali Sri Vyshnavi Devi
Roll-No: 200010023

Student 2: Meghana Sripalle
Roll-No: 200010028

1 Checking Home Directory File System

In this assignment, mfs is the file system that is being studied. In minix **/home** is of type **mfs** which can be verified using the command **cat /etc/fstab**. The required file system data corresponding to each folder is present in the fstab file of etc folder. With this trace, only the file system mounted at **/home** must be taken care of when working on the mfs file system.

```
# cat /etc/fstab
/dev/c0d0p0s0   /       mfs     rw                  0       1
/dev/c0d0p0s2   /usr    mfs     rw                  0       2
/dev/c0d0p0s1   /home   mfs     rw                  0       2
none            /sys    devman  rw,rslabel=devman   0       0
```

Figure 1: Checking File System Of **/home** Directory

In the Minix file system, inodes are used to store information about each file and directory on the file system. An inode, short for "index node," is a data structure that contains metadata about a file or directory, such as the file's size, permissions, ownership, and timestamps.
In Minix, each file or directory is represented by a unique inode number, which is used to access the corresponding inode in the file system. The inode contains information about the file's location on the disk, its permissions, and other metadata. The first 10 inodes in the Minix file system are reserved for the root directory and system files. The remaining inodes are used to represent user files and directories. Inodes are typically allocated when a file is created, and deallocated when the file is deleted.

2 Implementing Immediate Files

Immediate files are a type of file that contains the file data directly in the inode (index node), rather than in a separate data block. This can be useful for small files that don't require a lot of storage space. In Minix, the inode structure is 64 bytes in size. For a newly created file 40 bytes of the inode is free and for this assignment 32 bytes of it is being used to store the file data instead of storing it on a disk block and wasting disk space (internal fragmentation).

Figure 2: Inode Structure

2.1 Modified Files

2.1.1 /minix/include/minix/const.h

In this file, Immediate flag, similar to regular flag, is introduced.

```
#define I_REGULAR 0100000       /* regular file, not dir or special */
#define IMMEDIATE_FLAG 0110000  // flag for immediate file - lab 10
```

Figure 3: Changes made in const.h

2.1.2 /minix/lib/libc/gen/fslib.c

In this file, if the mode is not one of those mentioned in the function fs_mode_to_type(mode_t mode), we return the DT_REG flag; that is, the function does not recognise the immediate mode and cribs by aborting the operation. So that the file system runs efficiently, we return the DT_REG flag when we notice that the mode may be immediate in order to get around this.

2.1.3 /minix/servers/vfs/open.c

In the function common_open(), we made changes. The changes were made in such a way that when a new vnode is created, the omode value is set to IMMEDIATE_FLAG. The value is set so that the file is initially created as an immediate file.

```
if (S_ISREG(mode))
    return DT_REG;
else if (S_ISDIR(mode))
    return DT_DIR;
else if (S_ISLNK(mode))
    return DT_LNK;
else if (S_ISCHR(mode))
    return DT_CHR;
else if (S_ISBLK(mode))
    return DT_BLK;
else if (S_ISFIFO(mode))
    return DT_FIFO;
else if (S_ISSOCK(mode))
    return DT_SOCK;
else if ((mode & 0170000) == 0110000)
    return DT_REG;
```

Figure 4: Changes made in fslib.c

```
omode = IMMEDIATE_FLAG | (omode & ALLPERMS & fp->fp_umask);
vp = new_node(&resolve, oflags, omode);

r = err_code;
if (r == OK)
{
    exist = FALSE; /* We just created the file */
    struct vmnt *path_struct;
    path_struct = find_vmnt(vp->v_fs_e);
    char *directory_home = "/home";
    if (!strcmp(directory_home, path_struct->m_mount_path))
    {
        printf("file created: %llu\n", vp->v_inode_nr);
    }
}
```

Figure 5: Changes made in open.c

2.1.4 /minix/servers/vfs/link.c

While truncating, we checked if mode is 0110000 or not if it is then we are returning EINVAL else truncate the file to new size provided.

```
if (((vp->v_mode & S_IFMT) == 0110000) || (!S_ISREG(vp->v_mode) && !S_ISFIFO(vp->v_mode)))
    return (EINVAL);
```

Figure 6: Changes made in link.c

2.1.5 /minix/servers/vfs/read.c

In this file, we included some lines of code to make sure when some content is written/read to/from the file the inode number, bytes read/written and offset are printed and also we check if the directory of inode is ”/home” or not.

```
struct vmnt *path_struct;
path_struct = find_vmnt(vp->v_fs_e);
char *directory_home = "/home";
if (rw_flag == READING && !strcmp(path_struct->m_mount_path, directory_home))
{
    printf("file read: %llu; nbytes = %zu; offset = %llu\n", vp->v_inode_nr, size, f->filp_pos);
}

if (rw_flag == WRITING && !strcmp(path_struct->m_mount_path, directory_home))
{
    printf("file written: %llu; nbytes = %zu; offset = %llu\n", vp->v_inode_nr, size, f->filp_pos);
}
```

Figure 7: Changes made in read.c

2.1.6 /minix/fs/mfs/read.c

In read.c file we changed the function fs_readwrite() function in which we are checking if the current mode is immediate that is *mode_word == I_IMMEDIATE*, if rw_flag is set to WRITING then we check if position + nrbytes ¿ 32 or not that is the file size is going out of 32 bytes or not. In case it is not going out of 32 then we keep the variable imm_flag to 1 else it is a regular file so we copy the content stored in the inode to some buffer array, mark all zones as empty, change inode size, update time of inode, mark the inode as dirty and we request a new block and copy the buffer array content to block data field, mark the block as dirty, update position and file size and set inode mode to REGULAR and imm_flag to 0 as the file is no more an immediate file rather it is now a regular file. If the rw_flag is not set to WRITING then no change is required we set imm_flag to 1. Now, to handle reading and writing in immediate files, we calculate zone_position in the disk.

2.1.7 /minix/fs/mfs/write.c

In write.c if the file mode is immediate then we return NOBLOCK as there are no data blocks allotted to a immediate file instead they are stored in the inode itself.

```
if (IMMEDIATE_FLAG == (rip->i_mode & I_TYPE))
{
    return (NO_BLOCK);
}
```

Figure 8: Changes made in write.c

2.2 File Creation

Here, the file is created using touch command and so the file created message along with inode number is printed whch can be seen form the below picture.
command to create a file - touch new.txt

```
Minix: PID 357 created
assignment10 assignment22 assignment31 assignment51 meghana
assignment2  assignment23 assignment32 assignment9
Minix: PID 357 exited
# touch new.txt
Minix: PID 358 created
Minix3: File Created: 120
Minix: PID 358 exited
#
```

Figure 9: Creating file

2.3 File Reading

Here, the file is being read using the cat command. We can see that it is an immediate file from the output. Also the file contents are being displayed as well. Also give message of immediate file if the file is not regular. Also if the file is not immediate then the immediate file message is not printed.
command to read a file - cat new.txt

```
# cat new.txt
Minix: PID 359 created
Minix3: Reading from Immediate File.
Minix3: File Contentsof Immediate File:
Hello
Minix3: EOF - Immediate File
Minix3: File Read: 120; nbytes = 4096; offset = 0
Minix: PID 359 exited
```

Figure 10: reading immediate file

```
# cat new.txt
Minix: PID 362 created
Minix3: File Read: 120; nbytes = 4096; offset = 41
lessthan32lessthan32lessthan32lessthan32
Minix3: File Read: 120; nbytes = 4096; offset = 41
Minix: PID 362 exited
#
```

Figure 11: reading regular file

2.4 File Writing

Here, the file is being written using the echo command. We can see that the first file is an immediate file when its size is less than 32 bytes. When the size grows, we see that it is a regular file and immediate file message is not being printed.
command to write to a file - echo lessthan32 > new.txt

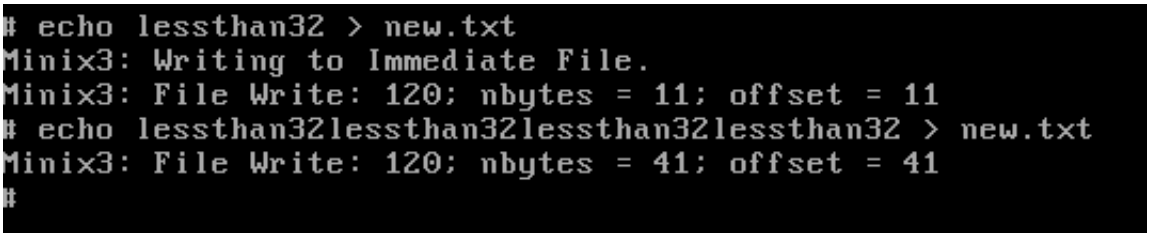

Figure 12: writing to a file

2.5 File Deletion

Here, when file deleted there is no print message and the output is as shown in below. command to delete - rm new.txt

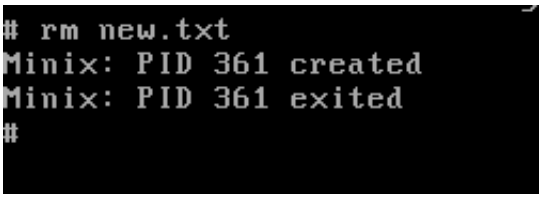

Figure 13: deleting file