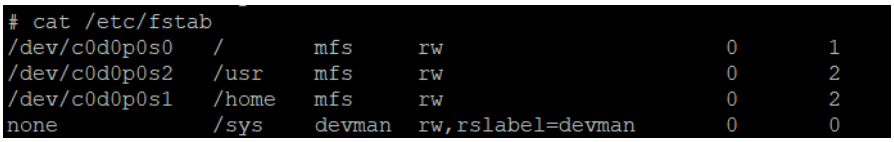
CS 314 – Operating Systems Lab
Lab–9 Report

Student Name: Kavali Sri Vyshnavi Devi **Roll-No:** 200010023

1 Assignment Description

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

1.1 Creating File

To print a message when a new file is created one need to change **common_open()** function in **open.c** which is present at the location **/usr/src/minix/servers/vfs**.

The following statements are added in the **common_open()** function.

```
struct vmnt *path_struct;
path_struct = find_vmnt(path_struct->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);
}
```

Here, a variable path_struct is created of type vmnt to get the mount path and after m_mount_path attribute of path_struct is used to compare if the directory is /home as it is mfs file system on which we need to work on. Here, to get m_mount_path, endpoint attribute - FS process' endpoint number (v_fs_e) of vp which is a pointer of struct node is passed to find_vmnt function (present in the file 'vmnt.c'). After, checking the directory is same as '/home' directory or not, file created message is printed along with the inode number of the file.

Here, in the print statement v_inode_nr attribute of vp is being printed which is of type ino_t which is defined in file 'vnode.h' and also the type of ino_t is uint64_t so used %llu in the print statement.

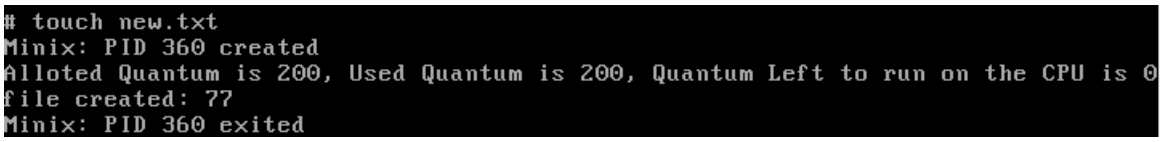


Figure 2: creating new file

From the above figure, on creating a newfile using touch command, file created message along with inode number is printed.

1.2 Reading File

To print a message when a file is read one need to change **read_write()** function in **read.c** which is present at the location **/usr/src/minix/servers/vfs**.

The following statements are added in the read_write() function.

```
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);
}
```

Here, a variable path_struct is created of type vmnt to get the mount path and after m_mount_path attribute of path_struct is used to compare if the directory is /home as it is mfs file system. Here, to get m_mount_path endpoint attribute (v_fs_e) of vp which is a pointer of struct node is passed to find_vmnt function. After, checking the directory file read message is printed along with the inode number on its (minor) device, size of the file and the offset which is a pointer to the place in the file where the next read or write will start. Also rw_flag is checked whether it is set to READING to print the message accordingly. Here, along with inode number size is also being printed which is of type size_t which can be verified by checking the signature of the function for which %zu format specifier is used and filp_pos attribute of variable f is also printed which represent offset of type off_t for which format specifier is %zu for decimal.
From the above figure, on reading a file created before using cat command, file read message along with inode number, size and offset are printed.

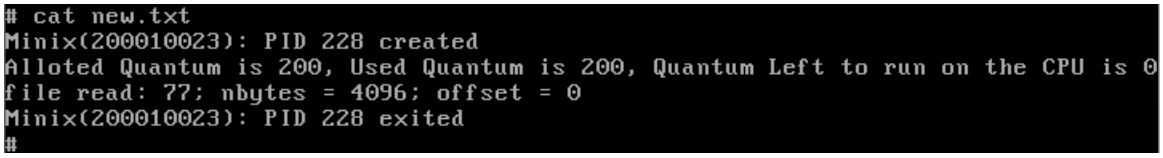


Figure 3: reading a file

1.3 Writing To File

To print a message when something is written to a file one need to change **read_write()** function in **read.c** which is present at the location **/usr/src/minix/servers/vfs**.

The following statements are added in the read_write() function.

```
struct vmnt *path_struct;
path_struct = find_vmnt(vp->v_fs_e);
char *directory_home = "/home";
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);
}
```

Here, a variable path_struct is created of type vmnt to get the mount path and after m_mount_path attribute of path_struct is used to compare if the directory is /home as it is mfs file system. Here, to get m_mount_path endpoint attribute (v_fs_e) of vp which is a pointer of struct node is passed to find_vmnt function. After, checking the directory file write message is printed along with the inode number on its (minor) device, size of the file and the offset which is a pointer to the place in the file where the next read or write will start. Also rw_flag is checked whether it is set to WRITING to print the message accordingly. Here, along with inode number size is also being printed which is of type size_t which can be verified by checking the signature of the function for which %zu format specifier is used and filp_pos attribute of variable f is also printed which represent offset of type off_t for which format specifier is %zu for decimal.

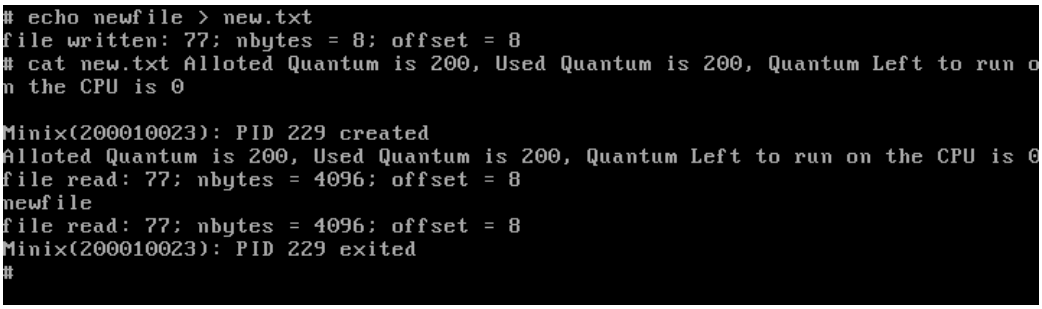


Figure 4: writing to a file

From the above figure, on writing to a file using echo newfile > new.txt command, file write message along with inode number, size and offset are printed.

1.4 Deleting a File

To print a message when a file is delete one need to change **do_unlink()** function in **link.c** which is present at the location **/usr/src/minix/servers/vfs**.

The following statements are included in the do_unlink() function.

```
lookup_init(&stickycheck, resolve.l_path, PATH_RET_SYMLINK, &vmp2, &vp);
stickycheck.l_vnode_lock = VNODE_READ;
stickycheck.l_vmnt_lock = VMNT_READ;

vp = advance(dirp, &stickycheck, fp);
char *directory_home = "/home";
if (!strcmp(vmp->m_mount_path, directory_home))
{
  printf("file deleted: %llu\n", vp->v_inode_nr);
}

if (vp != NULL)
{
  unlock_vnode(vp);
  put_vnode(vp);
}
```

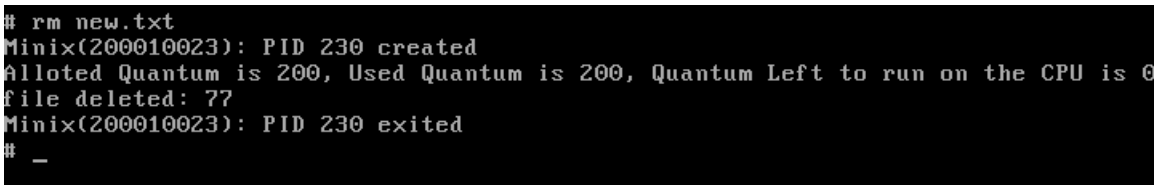


Figure 5: deleting a file

From the above figure, on deleting a file using rm command, file deleted message along with inode number are printed. Here, to get the inode number of the file being deleted advance function is being used for which given a directory, a component of a path are given as arguments and then the function looks up the component in the directory, find the inode, open it, and return a pointer to its inode slot. Also after printing the delete message reference counts to the file have to reduced to 0 for which unlock_vnode and put_vnode functions are being called, also lookup_init function is called before to initialize the vp variable to NULL as it is later used to store the result from advance.

2 Building The Updated Code

The updated files after editing are sent to guest OS using scp command from host OS.
Command - scp -P 2222 link.c open.c read.c root@localhost:/usr/lab9

```
C:\Users\DELL\Desktop\new>scp -P 2222 link.c open.c read.c root@localhost:/usr/lab9
root@localhost's password:
link.c                                                                    100%   17KB   3.4MB/s   00:00
open.c                                                                    100%   21KB   4.3MB/s   00:00
read.c                                                                    100%   11KB   4.1MB/s   00:00
```

Figure 6: copying files from host to guest os

To copy the edited files to the folder **/usr/src/minix/server/vfs** and build the OS after updating the code the following bash script is executed.

```
echo "copying files"
cp -f open.c /usr/src/minix/servers/vfs/open.c
cp -f link.c /usr/src/minix/servers/vfs/link.c
cp -f read.c /usr/src/minix/servers/vfs/read.c
echo "going to src directory and building the updated code"
cd /usr/src
make build MKUPDATE=yes
echo "build completed successfully"
exit 0
```

On successful build after making changes in link.c, read.c and open.c, rebooting the changes will be reflected in the minix operating system.

The below figures show the building of OS and it's successful completion.

```
# bash a.sh
copying files
going to src directory and building the updated code
```

Figure 7: Executing run.sh file

```
dependall ===> usr.bin/tsort
dependall ===> usr.bin/tty
dependall ===> usr.bin/ul
dependall ===> usr.bin/uname
dependall ===> usr.bin/unexpand
dependall ===> usr.bin/unifdef
dependall ===> usr.bin/uniq
dependall ===> usr.bin/units
dependall ===> usr.bin/unvis
dependall ===> usr.bin/unzip
dependall ===> usr.bin/users
dependall ===> usr.bin/uudecode
dependall ===> usr.bin/uuencode
dependall ===> usr.bin/uuidgen
dependall ===> usr.bin/vis
dependall ===> usr.bin/w
dependall ===> usr.bin/wall
dependall ===> usr.bin/wc
dependall ===> usr.bin/what
```

Figure 8: building os

```
Build started at:  Wed Mar 22 07:32:06 IST 2023
Build finished at: Wed Mar 22 07:38:24 IST 2023
```

Figure 9: successful build

```
# reboot
```

Figure 10: reboting the os