

Assignment 2

Niramay Vaidya 111605075
Srishti Shelke 111603056

1. Create a file named “f1” in directory “D1” and user 1 with appropriate access right to it. Create a link “f2” to “f1” in directory “D2” and give user 1 with appropriate access right to it. Demonstrate that both users can work on same file from different directories.

(Referred from the symbolic link documentation in the actual submission)

...Same behaviour is observed even in case of creating a symbolic link i.e. both files can be opened exactly as above simultaneously for editing and changes in one are immediately reflected in the other.

For both cases i.e. hard link and symbolic link, refer to video uploaded in current folder demonstrating the above behaviour.

(1_hard_symbolic_link_representation)

2. A function realpath() resolves all symbolic links in path and returns the *ultimate* target. Write a program to simulate realpath() to list the ultimate target of the only filename that are symbolic links in a directory. The program takes one optional argument, which is the name of a directory to be searched for the links. When no argument is specified, the search is conducted in the current working directory. Display appropriate error messages. Demonstrate by creating symbolic links A->B->C etc.

According to the actual submission, two scenarios for each case have been shown.

In the first case, given the name of a symbolic link, resolve it. This has been tested using two symbolic link files f3_start and f3_start_different. The tree command screenshot shows the file link structure of how the symbolic links have been created. This particular tests the realpath function on resolving symbolic links that not only point to files in subdirectories down the file tree but up the file tree too.

In the second case, when no symbolic link filename is provided as argument to the realpath code, it resolves all symbolic links present in the current directory. This functionality was tested on the same two files as mentioned above.

TODO- Full paths of the resolved files are not being printed currently, only their names.

3. Create a shared directory for usage with a purpose that any user (not super user) can create new files in this directory, but only the owner can delete his own files and everyone else can read all files. Demonstrate the functionality.

Executed this sequence of commands-

```
sudo -u \#0 mkdir test_dir
(onwer and group is root)
sudo chmod +t test_dir
cd test_dir
vi test_1
(owner and group is niramay)
sudo -u \#1001 vi test_2
(owner and group is temp)
sudo -u \#1002 vi test_3
```

(owner and group is test)

By doing this, every user can read other's files but can't write to them.

Any user can create a file and only a file's owner can delete that file, no-one else.

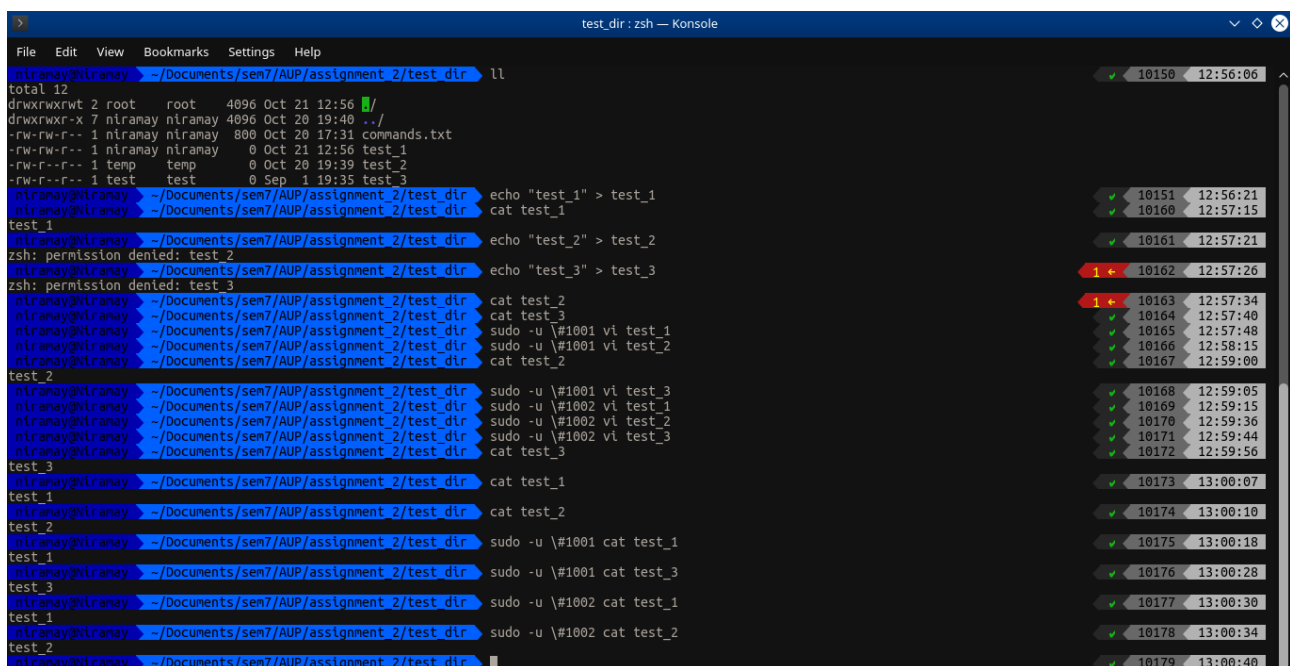
When the sticky bit set, only the owner of a file/directory can delete. Used on directories, it indicates that in order to rename or delete a file in that directory a user must both have write permissions to the directory and one of the following must be true-

user owns the file

user owns the directory

user is the superuser

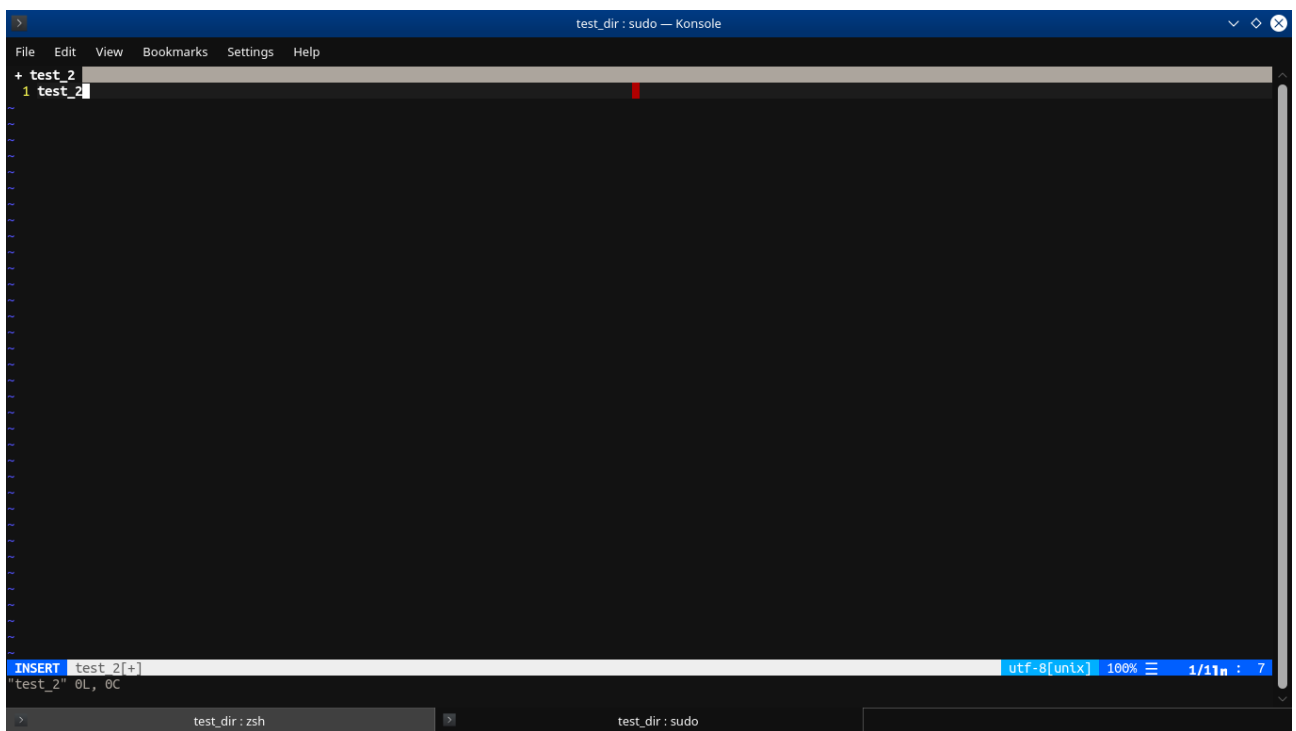
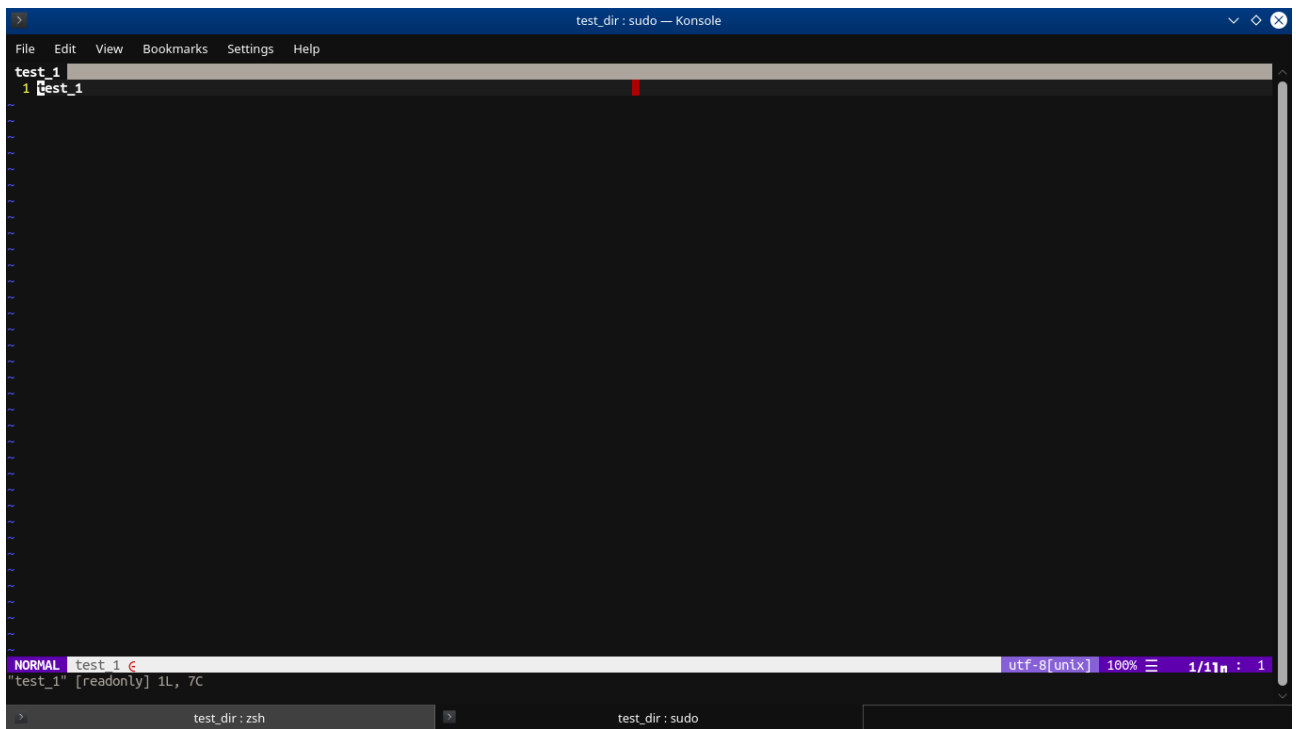
As a result, all users other than superuser get the write permissions to the directory via the others permissions of test_dir and as the sticky bit is set, users can only rename/delete those files which are owned by them (satisfying the first condition, none of these users own the test_dir directory) but can read contents of files owned by other users.

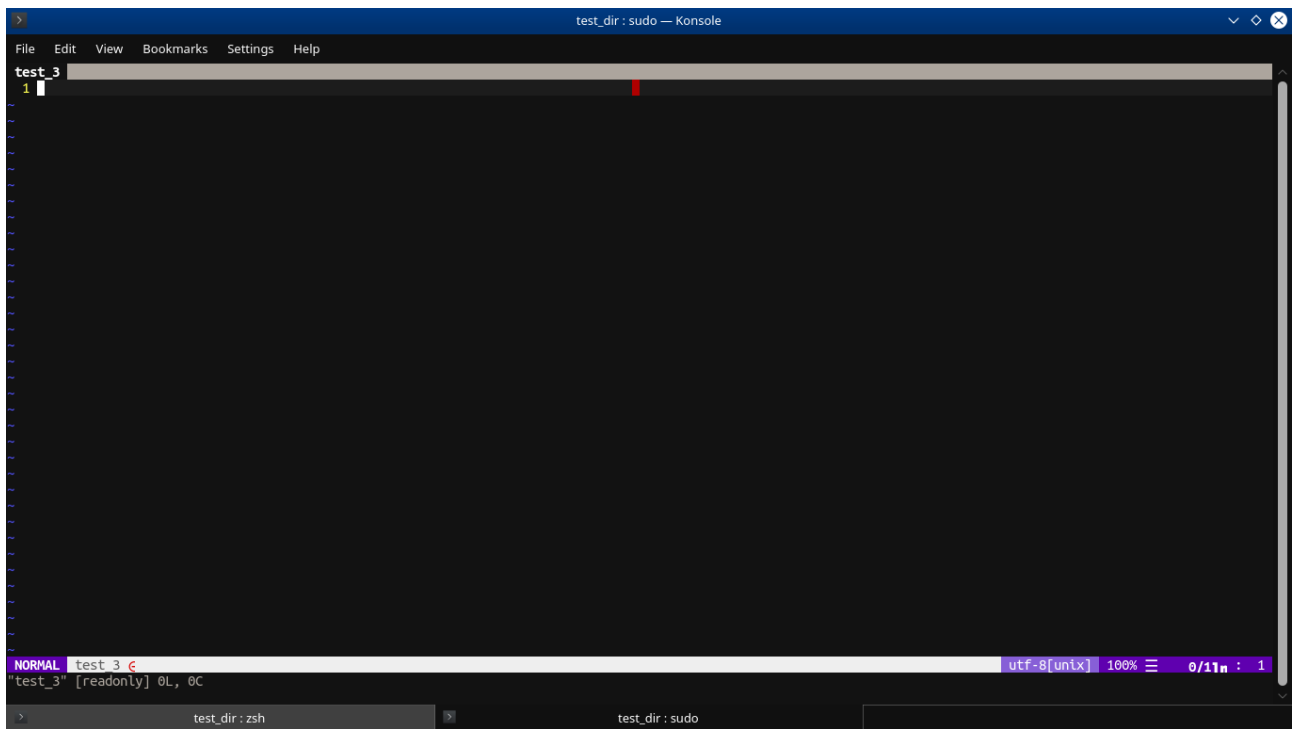


The screenshot shows a terminal window titled 'test_dir : zsh — Konsole'. The directory listing shows files owned by root and niramay, with permissions that include the sticky bit (t) for directories. Subsequent commands show users attempting to delete files, with 'permission denied' messages for files not owned by them. Then, users use 'sudo' to become other users (1001, 1002) and attempt to delete files, again failing for files not owned by them. Finally, users use 'vi' to edit files, successfully editing their own files but being denied access to others' files.

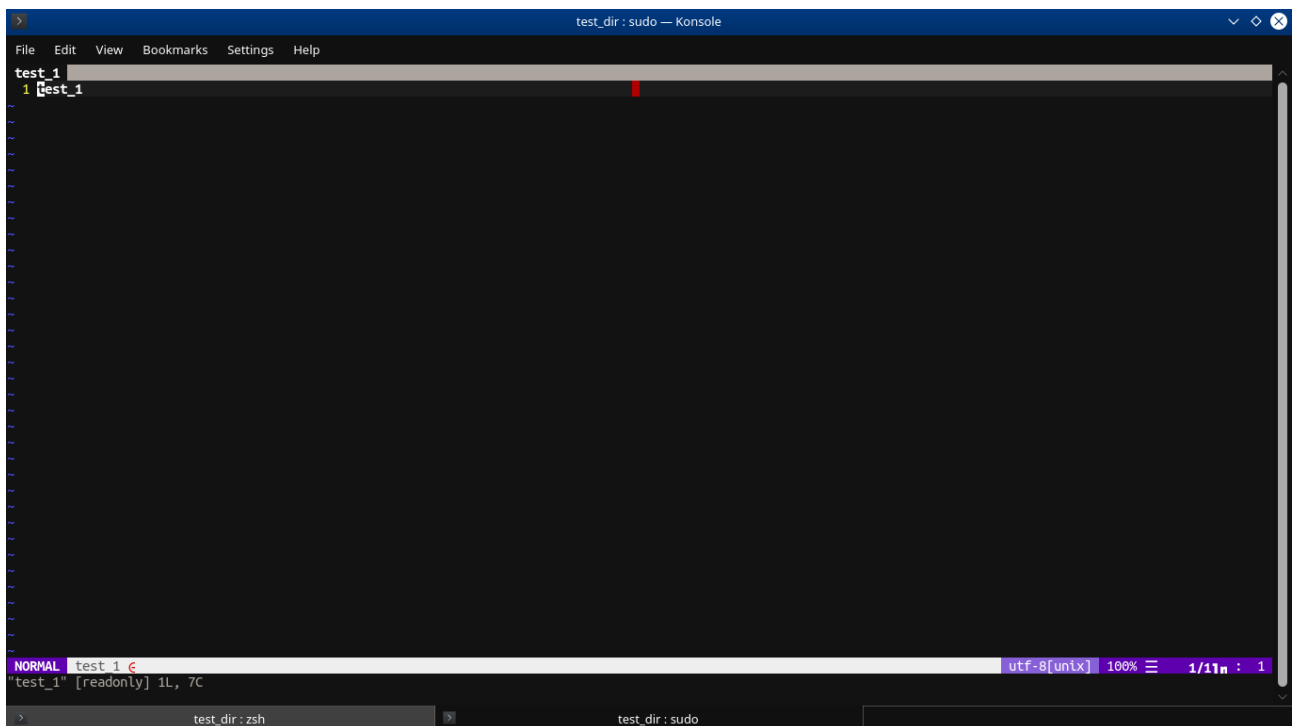
```
total 12
drwxrwxrwt 2 root  root  4096 Oct 21 12:56 /
drwxrwxr-x 7 niramay niramay 4096 Oct 20 19:40 ../
-rw-rw-r-- 1 niramay niramay 800 Oct 20 17:31 commands.txt
-rw-rw-r-- 1 niramay niramay  0 Oct 21 12:56 test_1
-rw-rw-r-- 1 temp  temp  0 Oct 20 19:39 test_2
-rw-rw-r-- 1 test  test  0 Sep  1 19:35 test_3
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir ll
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir echo "test_1" > test_1
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir cat test_1
test_1
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir echo "test_2" > test_2
zsh: permission denied: test_2
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir echo "test_3" > test_3
zsh: permission denied: test_3
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir cat test_2
test_2
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir cat test_3
test_3
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir sudo -u \#1001 vi test_1
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir cat test_2
test_2
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir sudo -u \#1001 vi test_3
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir cat test_3
test_3
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir sudo -u \#1002 vi test_1
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir cat test_2
test_2
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir sudo -u \#1002 vi test_2
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir cat test_3
test_3
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir sudo -u \#1002 vi test_3
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir cat test_1
test_1
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir cat test_2
test_2
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir sudo -u \#1001 cat test_1
test_1
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir sudo -u \#1001 cat test_3
test_3
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir sudo -u \#1002 cat test_1
test_1
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir sudo -u \#1002 cat test_2
test_2
niranay@niranay: ~/Documents/sem7/AUP/assignment_2/test_dir
```

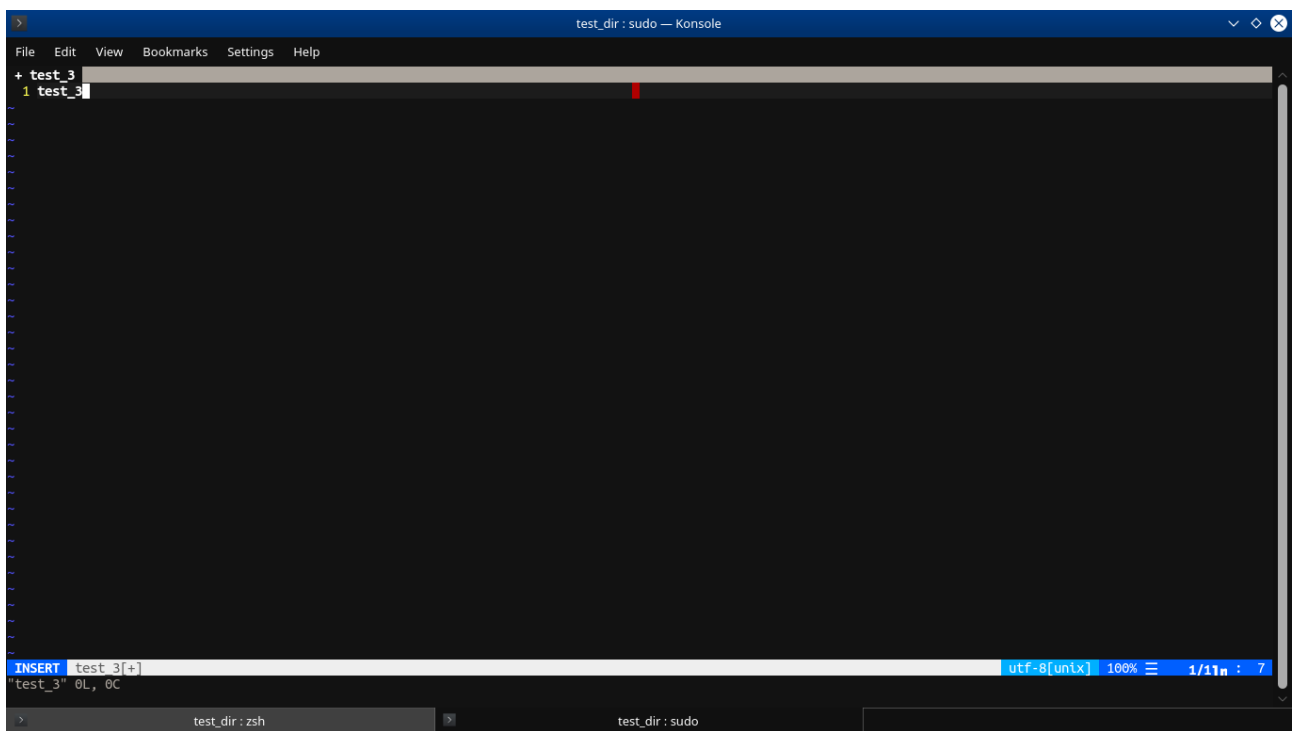
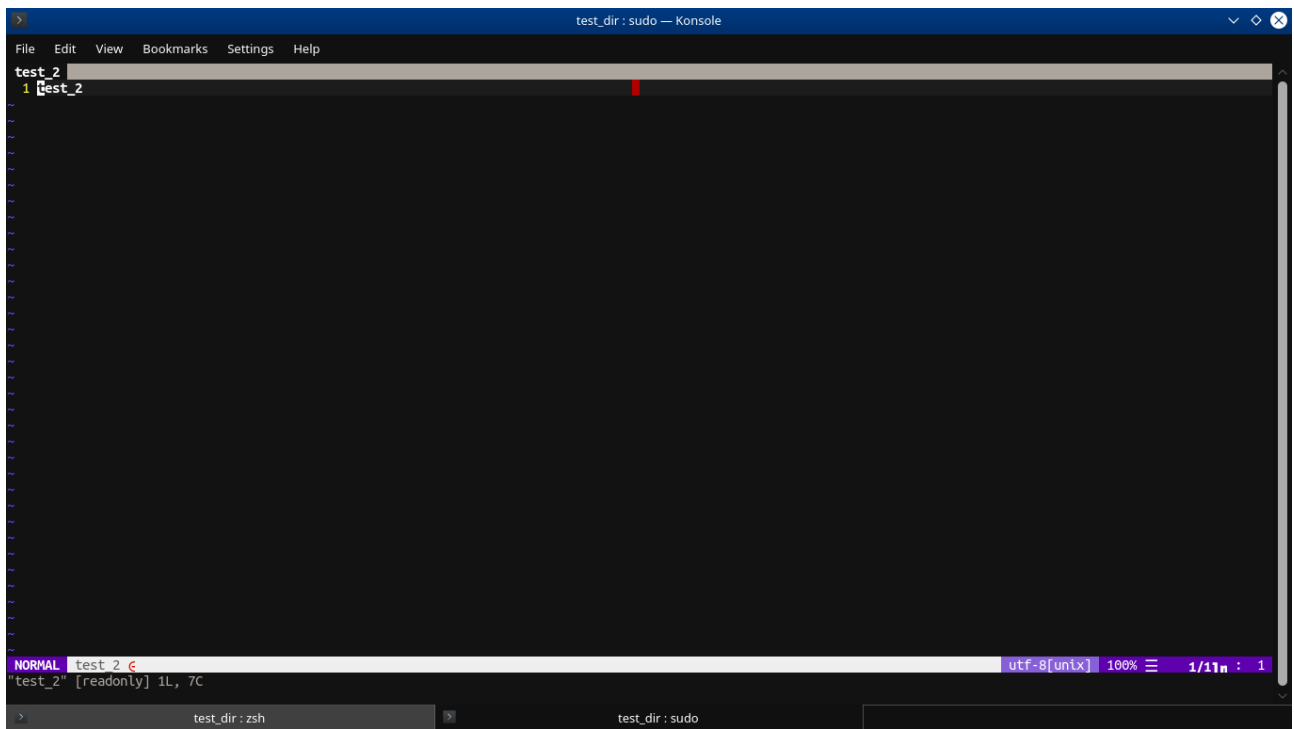
Opening files using vi as user temp (1001). This user can edit file owned by him i.e. test_2 but files test_1 and test_3 are readonly-





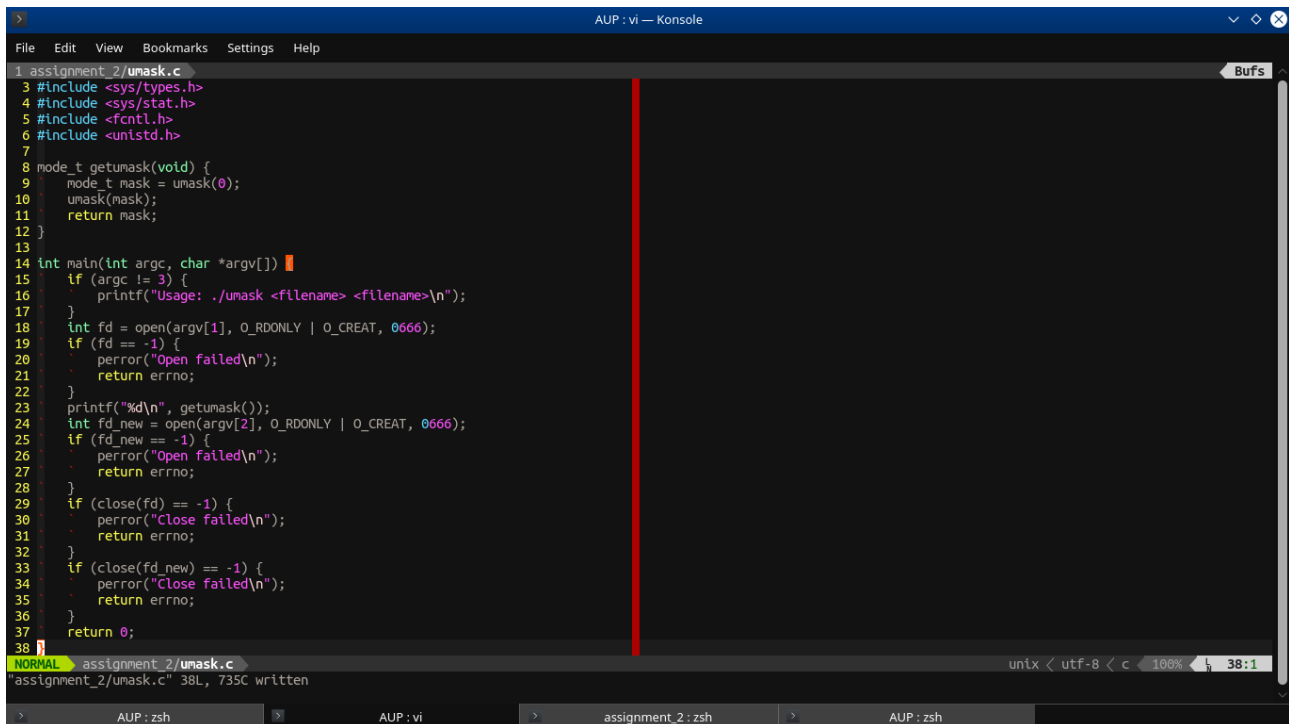
Opening files using vi as user test (1002). This user can edit file owned by him i.e. test_3 but files test_1 and test_2 are readonly-





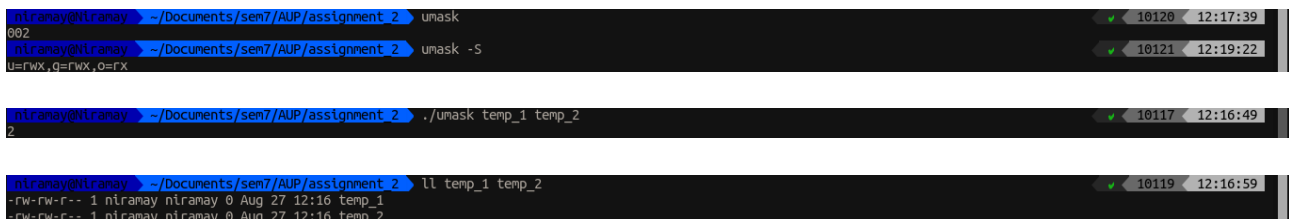
4. `umask()` always sets the process umask and, at the same time, returns a copy of the old umask. How can we obtain a copy of the current process umask while leaving it unchanged? Write a program to demonstrate.

Code-



```
1 assignment_2/umask.c
2
3 #include <sys/types.h>
4 #include <sys/stat.h>
5 #include <fcntl.h>
6 #include <unistd.h>
7
8 mode_t getumask(void) {
9     mode_t mask = umask(0);
10    umask(mask);
11    return mask;
12 }
13
14 int main(int argc, char *argv[]) {
15     if (argc != 3) {
16         printf("Usage: ./umask <filename> <filename>\n");
17     }
18     int fd = open(argv[1], O_RDONLY | O_CREAT, 0666);
19     if (fd == -1) {
20         perror("Open failed\n");
21         return errno;
22     }
23     printf("%d\n", getumask());
24     int fd_new = open(argv[2], O_RDONLY | O_CREAT, 0666);
25     if (fd_new == -1) {
26         perror("Open failed\n");
27         return errno;
28     }
29     if (close(fd) == -1) {
30         perror("Close failed\n");
31         return errno;
32     }
33     if (close(fd_new) == -1) {
34         perror("Close failed\n");
35         return errno;
36     }
37     return 0;
38 }
```

Output-



```
niranay@niranay: ~/Documents/sem7/AUP/assignment_2
$ umask
002
niranay@niranay: ~/Documents/sem7/AUP/assignment_2
$ umask -S
u=rwx,g=rwx,o=rwx

niranay@niranay: ~/Documents/sem7/AUP/assignment_2
$ ./umask temp_1 temp_2
2

niranay@niranay: ~/Documents/sem7/AUP/assignment_2
$ ll temp_1 temp_2
-rw-rw-r-- 1 niranay niranay 0 Aug 27 12:16 temp_1
-rw-rw-r-- 1 niranay niranay 0 Aug 27 12:16 temp_2
```

The current process umask is printed and is left unchanged before and after the call to `getumask` function i.e. the two files `temp_1` and `temp_2` created accordingly have the same permissions.