File system Internals

3c: File system Internals: stat, fstat, ustat, link/unlink,dup

Subject:- Unix Operating System Lab
Class :- TYIT
Name : Aditi Sudhir Ghate
PRN : **2020BTEIT00044**

3.c]Write a program to use link/unlink system call for creating logical link and identifying the difference using stat.

Objectives -

To learn about File system Internals.

Theory -

A) **link** - make a new name for a file

Syntax-

```
#include <unistd.h>
int link(const char *oldpath, const char *newpath);
```

link() creates a new link (also known as a hard link) to an existing file. If newpath exists it will not be overwritten.

This new name may be used exactly as the old one for any operation; both names refer to the same file (and so have the same permissions and ownership) and it is impossible to tell which name was the `original'.

On success, zero is returned. On error, -1 is returned, and errno is set appropriately. Hard links, as created by link(), cannot span filesystems. Use symlink() if this is required.

POSIX.1-2001 says that link() should dereference oldpath if it is a symbolic link. However, Linux does not do so: if oldpath is a symbolic link, then newpath is created as a (hard) link to the same symbolic link file (i.e., newpath becomes a symbolic link to the same file that oldpath refers to). Some other implementations behave in the same manner as Linux.

B) **unlink** - delete a name and possibly the file it refers to

Syntax-

```
#include <unistd.h>
int unlink(const char *pathname);
```

unlink() deletes a name from the filesystem. If that name was the last link to a file and no processes have the file open the file is deleted and the space it was using is made available for reuse.

If the name was the last link to a file but any processes still have the file open the file will remain in existence until the last file descriptor referring to it is closed. If the name referred to a symbolic link the link is removed. If the

name referred to a socket, fifo or device the name for it is removed but processes which have the object open may continue to use it.
On success, zero is returned. On error, -1 is returned, and errno is set appropriately.

Program-

```
#include<stdio.h>
#include<unistd.h>
#include<string.h>
int main()
{
 char old[100];
 char new[100];
 char ch;
 printf("Enter the old and new pathname: \n");
 gets(old);
 gets(new);
 int n = link(old,new);
 if(n==0)
 {
 printf("Linked successfully\n");
 }
 else
 {
 printf("Linked unsuccessfully\n");
 }
 printf("Do you want to unlink the new file?\n1:Y\n2:N\n");
 scanf("%c",&ch);
 if(ch=='Y'||ch=='y')
 {
 int m = unlink(new);
 if(m==0)
 {
 printf("Unlinked successfully\n");
 }
 else
 {
 printf("Unlinked unsuccessfully\n");
 }
 }
 else
 {
 printf("Not Unlinked\n");
 }
}
```

Output:-

```
aditi@aditi-Lenovo-ideapad-330S-14IKB-U:~/ADnOR/Assignments/3C$ gcc 3C.c
3C.c: In function ‘main’:
3C.c:10:2: warning: implicit declaration of function ‘gets’; did you mean ‘fgets
’? [-Wimplicit-function-declaration]
   10 |  gets(old);
      |  ^~~~
      |  fgets
/usr/bin/ld: /tmp/ccWmnXy9.o: in function `main':
3C.c:(.text+0x3d): warning: the `gets' function is dangerous and should not be u
sed.
aditi@aditi-Lenovo-ideapad-330S-14IKB-U:~/ADnOR/Assignments/3C$ ./a.out
Enter the old and new pathname:
aditi.txt
ghate.txt
Linked unsuccessfully
Do you want to unlink the new file?
1:Y
2:N
1
Not Unlinked
```

Conclusion:-

The concepts of creating link or shortcut to file and unlinking it understood through link and unlink function calls. The change in number of links takes place as we implement the program.