Bharatiya Vidya Bhavan's
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Autonomous Institute Affiliated to University of Mumbai)
Munshi Nagar, Andheri (W), Mumbai – 400 058.

Name: Adwait Purao
UID: 2021300101
Batch: B2
Experiment no.: 10

Aim: Write a program to prevent destructive update of files by locking as follows: Suppose the inode contains a new permission setting such that it allows only one process at a time to open the filefor writing, but many processes can open the file for reading.

Theory:

Internal Structure of UNIX File system

In this article, we will learn about the UNIX file systems and its internal structure in detail.

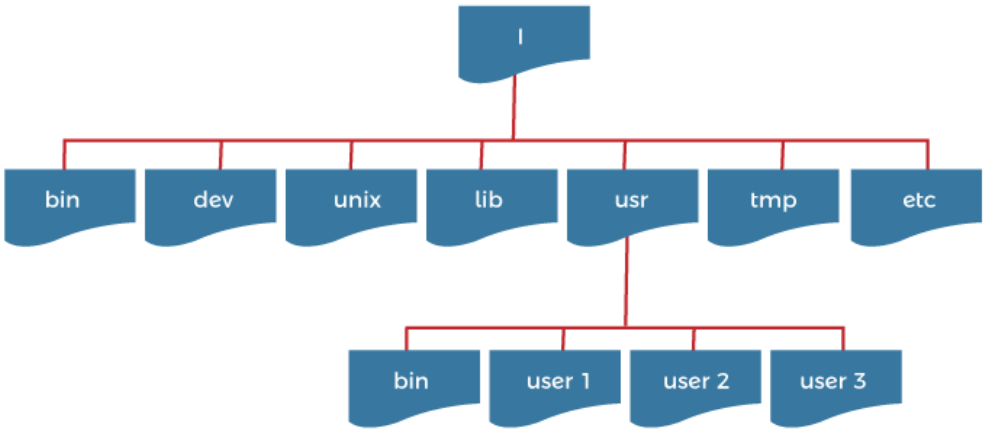
What do you mean by the UNIX Operating system?

All the utilities and applications are stored as files. It is a logical method of organizing and storing large amounts of information.

In UNIX, the file system is a hierarchical structure of files and directories where users can store and retrieve information using the files.

Structure of the UNIX file system

All the files in the UNIX system are related to one another.



UNIX file system

The Directory structure of the UNIX File system are given below:

Root (/): The top of the directory structure is called root. It is also called parent directory as it contains all the subdirectories of the UNIX file system. It is represented by using slash symbol in UNIX system.

/lib: This directory contains all the information of system libraries functions and some critical files such as kernel modules or device drivers. It also contains the system calls that the compiler can include in a program.

/bin: This directory contains the system's binary files and certain fundamental utilities. It is the directory for admin-level commands such as ls or cp. The path (var) always shows this directory in the list.

Following is the subdirectory of /bin directory:

- **/sbin:** If there is a command you cannot execute, the system administrator can execute this so that it could be in this directory. Only the system and administrator path show this directory.

/dev: It stands for "devices" and contains all the information about the device files used in the UNIX system. These files do not occupy any space on the disk. It

contains al; information about hard disks, file representations of peripheral devices, floppy disks, device information, and pseudo-devices.

Following is the subdirectory of /dev directory:

- **/dev /HD1:** This device file contains the first hard disk drive information.
- **/dev /HD2:** This device file contains the second hard disk drive information.
- **/dev /FD0:** This device file contains the first floppy disk drive information.
- **/dev /FD1:** This device file contains the second floppy disk drive information.

/etc : This directory and its subdirectories contain many of the UNIX configuration files and system databases. These files have many text files that can be changed according to the system's functionality. It also contains the information of your login name and password.

Following is the subdirectory of /etc directory:

- **/etc/passwd: This directory contains information about your system password.**
- **./etc/shadow: This directory contains information on the original password.**

/home: This directory contains all the information of home directories for the users. Whenever a user log in to the system, the UNIX system automatically places you in a home directory.

For Example:

If you log in the UNIX system using the login name demo, you would be in a directory that could have pathname /home/demo

This directory is created by the system when your account is opened. You can change your home directory using the cd command whenever you require. To check the home directory in which you are currently working, you can use the pwd command.

/tmp: This directory contains the information of all the temporary files created by UNIX or by the user in the UNIX system.

/var: The variable directory of a UNIX file system contains all the information of print jobs and outgoing and incoming emails. The variable part of a file system may be placed for files as database storage.

Following is the subdirectory of /var directory:

- **/var/log: This directory contains the information of system log files.**
- **/var/mail: This directory contains the information where all incoming emails are stored.**
- **/var/spool: This directory is also called spool directory. It contains information on print jobs, mail spools, and other queued tasks.**
- **/var/tmp: This directory contains the information of temporary files.**

/usr: This directory contains all the information related to the users in a UNIX system. It means it contains all the user account such as user1, user2, and so on.

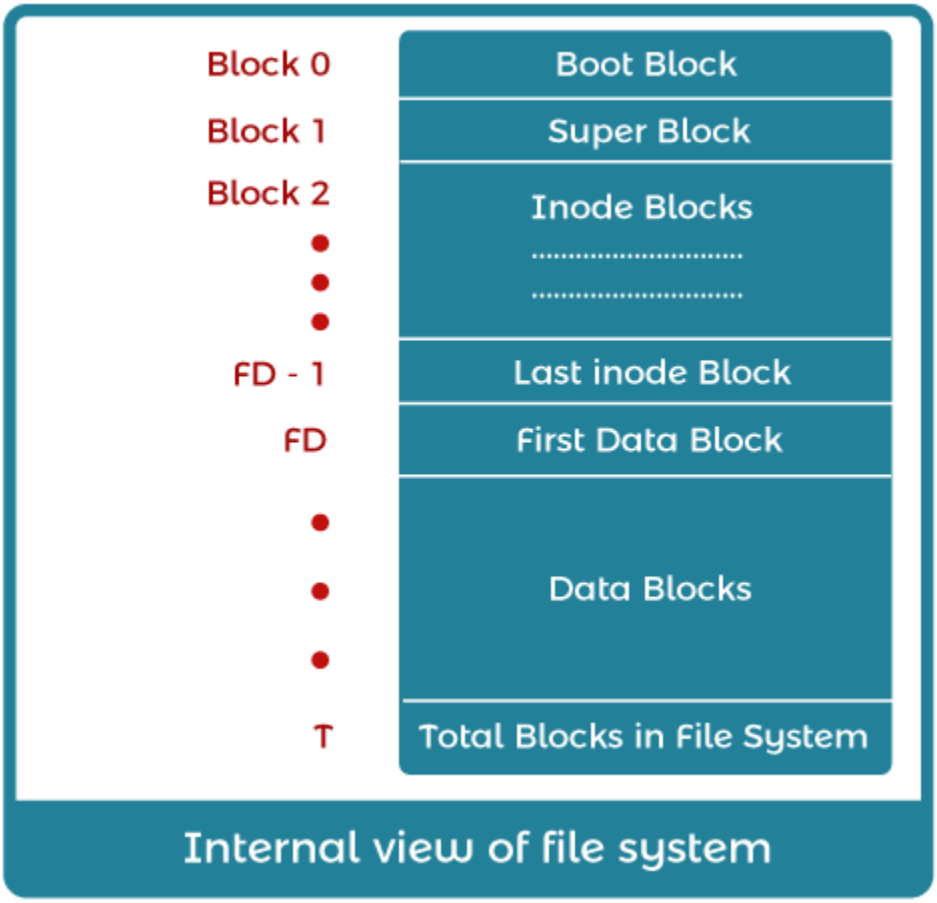
Following is the subdirectory of /usr directory:

- **/usr/bin: This directory stores the information of all the user binary used in UNIX system.**
- **/usr/include: This directory stores the information of all the development headers used in the system. The # mostly uses header files include directives in the C/C++ programming language.**

What do you mean by the internal structure of the UNIX file system?

A file system is a group of files that contains information about how files and inodes are stored on the disks.

The internal view of the UNIX file system shows in the following figure:



In the above figure, it defines several parameters such as:

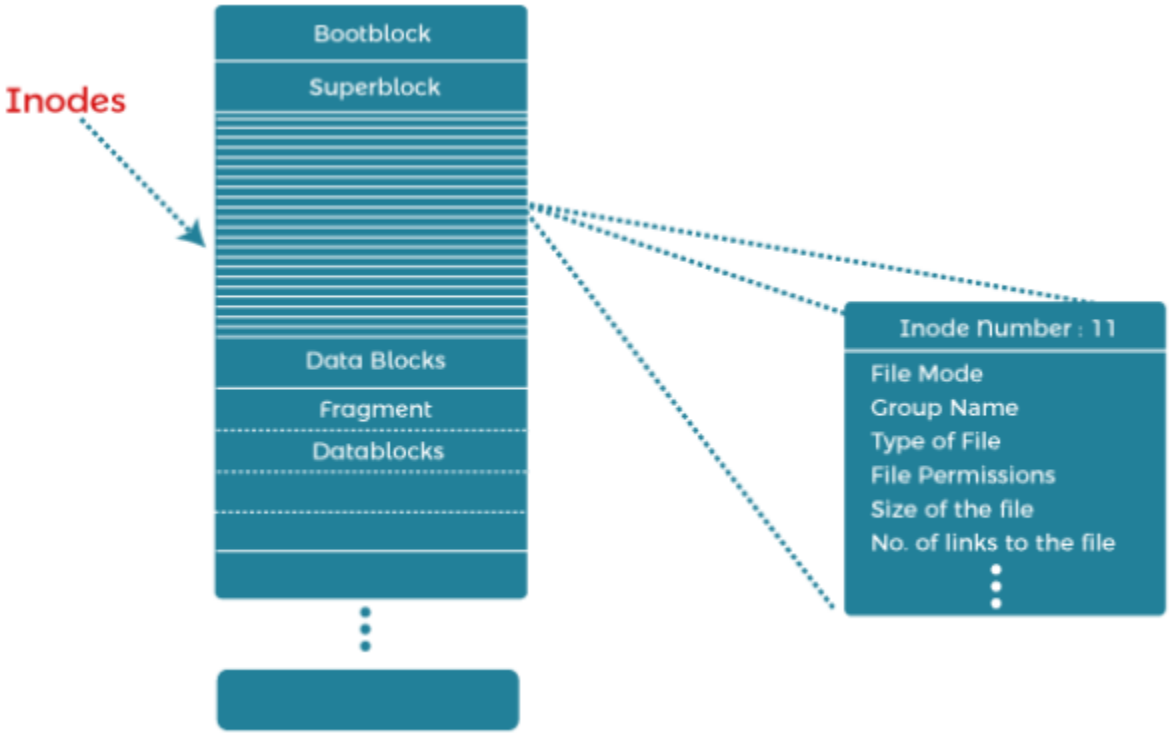
- **Boot block: It is the first block of the UNIX file system and contains a small bootstrap program, i.e., known as a bootstrap program. It is loaded into the main memory and executed when booted up. This bootstrap program is fetched from the boot block of one file system known as a root file system.**
- **Superblock: It contains static parameters of the file system like total size, the total number of data blocks and file system status, number of inodes, the free and used inodes, and the block size for the file system. The superblock is kept in the memory and maintained by the kernel. The file system organization divides disk partitions into one or more areas called cylinder groups. This group consists of one or more consecutive cylinders on a disk. Each cylinder consists of a duplicated**

copy of the superblock, space for inodes, and a piece of summary information describing the usage of data blocks within the cylinder group.

- **Inodes: It stands for index node. When a file is created, an inode is also created to keep the information about all attributes of the particular file. The number of inodes represents the maximum number of files in the UNIX system.**

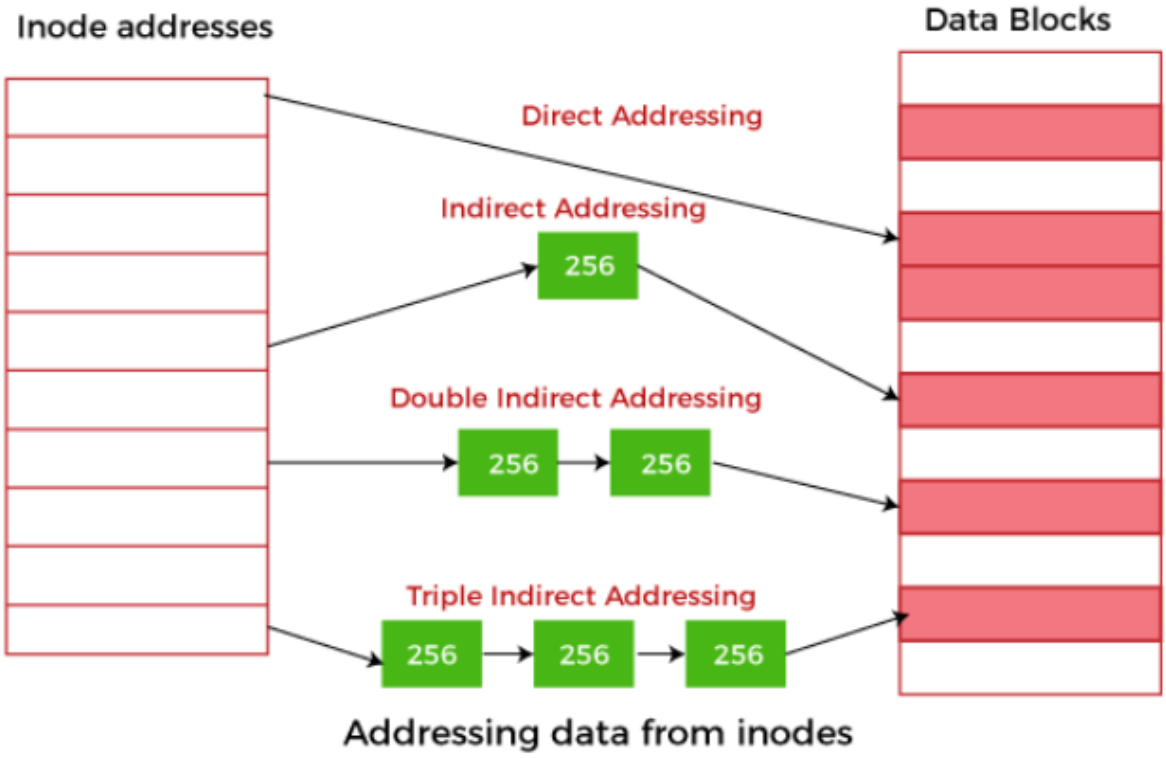
  A file's inode is responsible for storing the entire file's relevant data, except its name, which is stored as an entry in its parent directory.

  When a file is opened, the kernel copies its corresponding inode from disk to main memory. The inode includes the type of file, a file's access information, i.e., read, write or execute several links to the file, length of files in bytes, and representations of the user and group who owns the file.



In the above figure, a closer look at the inode structure reveals that the addressing is not always straightforward. In this, there is a concept of indirect,

double, and even triple indirect disk block address as shown in the below figure:



Addressing data from inodes

- **In UNIX, when a file is created, it is assigned a unique number known as an inode number. In this way, every file in UNIX has an inode number. UNIX treats all directories as files, so they also have an inode number.**

  An inode number assigned to a file can be accessed using the "ls- i" command, while the "ls- l" command will retrieve the inode information.
- **Data blocks: The data blocks start at the end of the inode list and contain the file data. An allocated data block can belong to one and only one file in the UNIX system. The disk head seeks to access the data block, and we allocate blocks from the same cylinder group. If a single file cannot take up all the blocks in a cylinder group and a file exceeding a certain size then a further block allocation redirected to a different cylinder group.**

Flock:

flock - apply or remove an advisory lock on an open file

SYNOPSIS

#include <sys/file.h>

int flock(int *fd*, int *operation*);

DESCRIPTION

Apply or remove an advisory lock on the open file specified by *fd*. The parameter *operation* is one of the following:

Tag	Description
LOCK_SH	Place a shared lock. More than one process may hold a shared lock for a given file at a given time.
LOCK_EX	Place an exclusive lock. Only one process may hold an exclusive lock for a given file at a given time.
LOCK_UN	Remove an existing lock held by this process.

A call to flock() may block if an incompatible lock is held by another process. To make a non-blocking request, include LOCK_NB (by *OR*ing) with any of the above operations.

A single file may not simultaneously have both shared and exclusive locks.

Locks created by flock() are associated with an open file table entry. This means that duplicate file descriptors (created by, for example, fork(2) or dup(2)) refer to the same lock, and this lock may be modified or released using any of these descriptors. Furthermore, the lock is released either by an explicit LOCK_UN operation on any of these duplicate descriptors, or when all such descriptors have been closed.

If a process uses open(2) (or similar) to obtain more than one descriptor for the same file, these descriptors are treated independently by flock(). An attempt to lock the file using one of these file descriptors may be denied by a lock that the calling process has already placed via another descriptor.

A process may only hold one type of lock (shared or exclusive) on a file. Subsequent flock() calls on an already locked file will convert an existing lock to the new lock mode.

Locks created by flock() are preserved across an execve(2).

A shared or exclusive lock can be placed on a file regardless of the mode in which the file was opened.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

Error Code	Description
EBADF	***fd* is not a not an open file descriptor.**

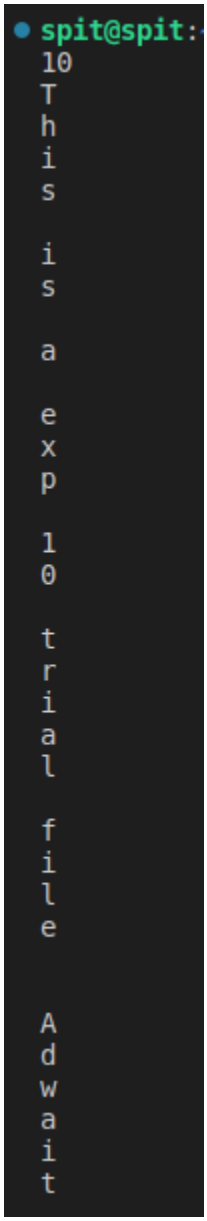
EINTR	**While waiting to acquire a lock, the call was interrupted by delivery of a signal caught by a handler.**
EINVAL	***operation* is invalid.**
ENOLCK	**The kernel ran out of memory for allocating lock records.**
EWOULDBLOCK	**The file is locked and the LOCK_NB flag was selected.**

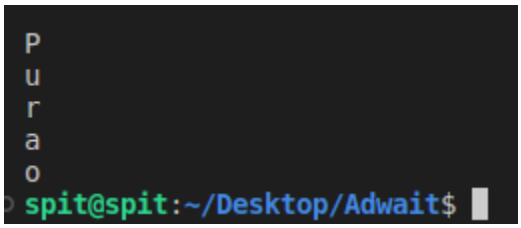
Code:

```
#include <stdio.h>
#include <sys/file.h>
#include <unistd.h>

int main()
{
   FILE *fp; // declare a file pointer variable named fp
   fp = fopen("/home/spit/Desktop/Adwait/trial.txt", "rw+"); // open a
file named trial.txt in read-write mode and assign the file stream to fp
   int fd = fileno(fp); // get the file descriptor of the file associated
with fp
   flock(fd, LOCK_SH); // obtain a shared lock on the file using the file
descriptor fd
   char c; // declare a character variable named c
   while (1) // enter an infinite loop
   {
       c = fgetc(fp); // read the next character from the file stream and
assign it to c
       if (feof(fp)) // if the end of the file has been reached
       {
           break; // break out of the loop
```

```
        }
        printf("%c\n", c); // print the character to the console followed
by a newline character
        sleep(1); // pause the program for 1 second
    }
    fclose(fp); // close the file stream associated with fp

    return 0;
}
```

Output:


spit@spit:
10
T
h
i
s
i
s
a
e
x
p
1
0
t
r
i
a
l
f
i
l
e
A
d
w
a
i
t


P
u
r
a
o
spit@spit:~/Desktop/Adwait$

Conclusion:

In conclusion, the program demonstrates how to use file I/O functions to read the contents of a file character by character, while also utilizing file locking to ensure that the file is not modified by other processes while the program is reading from it. The program also uses the concept of file descriptors to obtain the file descriptor associated with a file stream, and the concept of infinite loops to continuously read characters from the file until the end of the file is reached. Finally, the program uses character I/O to read and print individual characters from the file using the "fgetc" and "printf" functions, respectively.