**File locking system call**

**Subject - Unix Operating System**

**Name – Rachit Bhat**

**PRN – 22610007 Class – TYIT**

**Assignment No – 3B(a)**

**Title-** Write a program to lock the file using lockf system call. Check for mandatory  
locks, the file must be a regular file with the set-group-ID bit on and  
the group executes permission off. If either condition fails, all  
record locks are advisory.

**Objectives-**

1. To learn about File locking-mandatory and advisory locking.

**Theory-** File locking is a mechanism that restricts access to a file for the purpose of ensuring data consistency and preventing conflicting operations. File locks can either be mandatory or advisory:

* **Advisory locks**: These locks are respected only if the program explicitly checks for and honours them. They are not enforced by the kernel.
* **Mandatory locks**: These locks are enforced by the kernel, and if a file is locked, no other process can access it in a conflicting way.

The **lockf** system call is used to manage file locking in Unix-like operating systems. A file can be locked in a specific region or the entire file.

1. **lockf() System Call-** The lockf system call is used to apply advisory or mandatory locks on a file. Its syntax is:

int lockf(int fd, int cmd, off\_t len);

**fd**: The file descriptor of the file you want to lock.

**cmd**: The command that specifies the lock operation:

* **F\_LOCK:** Lock the file.
* **F\_ULOCK:** Unlock the file.
* **F\_TLOCK:** Try to lock the file; return immediately if the file cannot be locked.
* **F\_TEST:** Test whether the file is locked.

**len**: The number of bytes to lock, starting from the current position of the file descriptor. If this is set to 0, it locks the entire file.

1. **Mandatory Locking:** Mandatory locks are enforced by the operating system under specific conditions. For a file to use mandatory locking, the following conditions must be met:
2. The file must be a regular file.
3. The set-group-ID (SGID) bit must be set on the file.
4. The group execute permission must be turned off.

If these conditions are not satisfied, file locks will be advisory, meaning they will not be enforced by the kernel, and it is up to the program to respect the locks.

**Program:**

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/stat.h>

#include <errno.h>

int main() {

int fd;

struct stat fileStat;

// Open the file

fd = open("example.txt", O\_RDWR);

if (fd == -1) {

perror("Error opening file");

return 1;

}

// Get file status

if (fstat(fd, &fileStat) == -1) {

perror("Error getting file status");

close(fd);

return 1;

}

// Check if the file is a regular file, SGID is set, and execute permission for the group is off

if (S\_ISREG(fileStat.st\_mode) &&

(fileStat.st\_mode & S\_ISGID) &&

!(fileStat.st\_mode & S\_IXGRP)) {

// Apply mandatory locking using lockf system call

if (lockf(fd, F\_LOCK, 0) == -1) {

perror("Error locking file");

close(fd);

return 1;

}

printf("File locked successfully with mandatory locking.\n");

// Unlock the file when done

if (lockf(fd, F\_ULOCK, 0) == -1) {

perror("Error unlocking file");

close(fd);

return 1;

}

printf("File unlocked successfully.\n");

} else {

printf("Mandatory locking conditions not met. Using advisory locking.\n");

// Apply advisory locking (F\_TLOCK)

if (lockf(fd, F\_TLOCK, 0) == -1) {

perror("Error locking file");

close(fd);

return 1;

}

printf("File locked with advisory locking.\n");

// Unlock the file when done

if (lockf(fd, F\_ULOCK, 0) == -1) {

perror("Error unlocking file");

close(fd);

return 1;

}

printf("File unlocked successfully.\n");

}

// Close the file descriptor

close(fd);

return 0;

}

**Conclusion:** In this program, we demonstrate the usage of the lockf system call to lock and unlock files based on mandatory or advisory locking conditions. Understanding these locks is crucial in ensuring data integrity and preventing race conditions in multi-process applications.