**File locking system call**

**Subject - Unix Operating System**

**Name – Rachit Bhat**

**PRN – 22610007 Class – TYIT**

**Assignment No – 3B(c)**

**Title-** Write a program to lock file using fnctl system call

**Objective-**

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

**Theory-** File locking is essential to manage concurrent access to files by multiple processes, ensuring data integrity. The fcntl() system call is one way to implement file locking in Unix-like systems. Unlike flock(), which provides advisory locking, fcntl() can be used for both advisory and mandatory locking, depending on how it is configured.

There are two types of locking mechanism:

* **Advisory Locking**: This is a voluntary mechanism where processes can lock a file to prevent other processes from accessing it in conflicting ways. It is up to the processes to check for existing locks.
* **Mandatory Locking**: In mandatory locking, the operating system enforces the lock. A process cannot access a locked file unless the lock is released.

1. **fcntl System Call:** The fcntl() system call allows you to manipulate file descriptors and set file locks. To lock a file, the fcntl() function requires the use of flock structure that provides options to acquire or release locks.

The syntax is:

**int fcntl(int fd, int cmd, struct flock \*lock);**

**fd**: The file descriptor for the file to be locked.

**cmd**: The command to be executed, such as F\_SETLK, F\_GETLK, or F\_SETLKW.

* **F\_SETLK:** Set the lock.
* **F\_GETLK:** Get the status of the lock.
* **F\_SETLKW:** Set the lock and wait if it's blocked

**lock**: A pointer to a flock structure that describes the lock parameters (type, start, length, etc.).

The flock structure has the following fields:

* **l\_type:** Type of lock (can be F\_RDLCK for read lock, F\_WRLCK for write lock, or F\_UNLCK to unlock).
* **l\_whence:** The reference point for the lock (can be SEEK\_SET, SEEK\_CUR, or SEEK\_END).
* **l\_start:** The starting byte of the region to lock.
* **l\_len:** The length of the region to lock.
* **l\_pid:** The process ID of the process holding the lock.\

**Program-**

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <errno.h>

int main() {

// Open the file

int fd = open("example.txt", O\_WRONLY | O\_CREAT, S\_IRUSR | S\_IWUSR);

if (fd == -1) {

perror("Error opening file");

return 1;

}

// Set up the flock structure

struct flock lock;

lock.l\_type = F\_WRLCK; // Set the lock type to write lock

lock.l\_whence = SEEK\_SET; // Start from the beginning of the file

lock.l\_start = 0; // Lock the file from the start

lock.l\_len = 0; // Lock the entire file

lock.l\_pid = getpid(); // Set the lock's process ID

// Lock the file using fcntl

if (fcntl(fd, F\_SETLK, &lock) == -1) {

if (errno == EAGAIN || errno == EACCES) {

printf("The file is already locked by another process.\n");

} else {

perror("Error locking file");

}

close(fd);

return 1;

}

printf("File locked successfully. Press Enter to unlock.\n");

// Wait for user input to release the lock

getchar();

// Unlock the file

lock.l\_type = F\_UNLCK; // Change the lock type to unlock

if (fcntl(fd, F\_SETLK, &lock) == -1) {

perror("Error unlocking file");

close(fd);

return 1;

}

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

// Close the file

close(fd);

return 0;

}

**Conclusion-** The fcntl() system call offers a robust and flexible way to implement file locking in a program, allowing for both advisory and mandatory locking. Understanding how to use fcntl() is important when managing concurrent file access in multi-process applications.