**File locking in Linux:**

File locking in Linux is a mechanism used to prevent multiple processes or threads from simultaneously accessing or modifying the same file. It is a synchronization technique that ensures data consistency and integrity when multiple processes are working with shared files.

**Linux provides several methods for file locking, including:**

1. Advisory locks: Also known as "advisory file locking," this method allows processes to request a lock on a file using the **fcntl()** system call. Advisory locks do not prevent other processes from accessing the file; instead, they rely on cooperation between processes to honor the locks. Advisory locks are useful when all participating processes agree to respect the locks.
2. Mandatory locks: Also referred to as "mandatory file locking" or "MandLock," this method enforces locks on files by the kernel, making it mandatory for all processes to respect the locks. Mandatory locks are set using the **fcntl()** system call with the **F\_SETLK** command and the **F\_WRLCK** or **F\_RDLCK** flag to acquire a write or read lock, respectively.
3. POSIX record locks: POSIX record locks provide a way to lock specific regions within a file using the **fcntl()** system call with the **F\_SETLK** or **F\_SETLKW** command. These locks are associated with an open file descriptor and can be either shared (read) or exclusive (write). If a process tries to access a locked region, it can block (with **F\_SETLKW**) or fail (with **F\_SETLK**) depending on the lock type.
4. File lease: Introduced in the Linux 4.5 kernel, file leases are a type of advisory lock that allows processes to indicate their interest in a file. By using the **fcntl()** system call with the **F\_SETLEASE** command, a process can request a lease on a file, and other processes can be notified of lease changes. File leases are typically used for coordinating access to networked or distributed file systems.

These locking mechanisms provide different levels of coordination and enforcement for file access in Linux. It's important to choose the appropriate method based on the specific requirements and characteristics of your application.

Code:

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

 int file\_descriptor;

 struct flock lock;

 // Open the file

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

 if (file\_descriptor == -1) {

 perror("Failed to open file");

 exit(1);

 }

 // Set the lock structure

 lock.l\_type = F\_WRLCK; // Exclusive (write) lock

 lock.l\_whence = SEEK\_SET;

 lock.l\_start = 0; // Start locking from the beginning of the file

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

 // Try to acquire the lock

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

 perror("Failed to acquire lock");

 exit(1);

 }

 // Lock acquired successfully

 printf("Lock acquired. Press enter to release the lock.\n");

 getchar();

 // Release the lock

 lock.l\_type = F\_UNLCK;

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

 perror("Failed to release lock");

 exit(1);

 }

 // Close the file

 close(file\_descriptor);

 return 0;

}

This code opens the file "example.txt" and attempts to acquire an exclusive (write) lock on the entire file using fcntl(). If the lock is acquired successfully, it waits for the user to press enter to release the lock. Finally, it releases the lock and closes the file.

Note that this is a basic example, and in a real-world scenario, you would need to handle error conditions, implement proper synchronization mechanisms, and consider potential race conditions or deadlocks.