FILE AND RECORD LOCKING

UNIX systems allow multiple processes to read and write the same file concurrently which provides data sharing among processes. It also renders difficulty for any process in determining when data in a file can be overridden by another process.

In some of the applications like a database manager, where no other process can write or read a file while a process is accessing a database file. To overcome this drawback, UNIX and POSIX systems support a file locking mechanism.

File locking is applicable only for regular files. It allows a process to impose a lock on a file so that other processes cannot modify the file until it is unlocked by the process.

A process can impose a write lock or a read lock on either a portion of a file or an entire file.

The difference between write locks and read locks is that when a write lock is set, it prevents other processes from setting any overlapping read or write locks on the locked region of a file. On the other hand, when a read lock is set, it prevents other processes from setting any overlapping write locks on the locked region of a file.

The intention of a write lock is to prevent other processes from both reading and writing the locked region while the process that sets the lock is modifying the region. A write lock is also known as an _exclusive lock_.

The use of a read lock is to prevent other processes from writing to the locked region while the process that sets the lock is reading data from the region. Other processes are allowed to lock and read data from the locked regions. Hence, a read lock is also called a _shared lock_.

3.2.1 Mandatory Lock

Mandatory locks are enforced by an operating system kernel.

If a mandatory exclusive lock is set on a file, no process can use the _read_ or _write_ system calls to access data on the locked region.

If a mandatory shared lock is set on a region of a file, no process can use the _write_ system call to modify the locked region.

It is used to synchronize reading and writing of shared files by multiple processes: If a process locks up a file, other processes that attempts to write to the locked regions are blocked until the former process releases its lock.

Mandatory locks may cause problems: If a runaway process sets a mandatory exclusive lock on a file and never unlocks it, no other processes can access the locked region of the file until either the runaway process is killed or the system is rebooted.

System V.3 and V.4 support mandatory locks.

3.2.2 Advisory Lock

An advisory lock is not enforced by a kernel at the system call level.

This means that even though lock (read or write) may be set on a file, other processes can still use the *read* or *write* APIs to access the file.

To make use of advisory locks, processes that manipulate the same file must cooperate such that they follow this procedure for every read or write operation to the file:

a. Try to set a lock at the region to be accessed. If this fails, a process can either wait for the lock request to become successful or go do something else and try to lock the file again later.

b. After a lock is acquired successfully, read or write the locked region release the lock

c. The drawback of advisory locks are that programs that create processes to share files must follow the above file locking procedure to be cooperative. This may be difficult to control when programs are obtained from different sources.

All UNIX and POSIX systems support advisory locks.

UNIX System V and POSIX.I use the *fcntl* API for file locking. The prototype of the *fcntl* API is:

#include<fcntl.h>
int fcntl(int fdesc, int cmd_flag, …);

The *fdesc* argument is a file descriptor for a file to be processed. The *cmd flag argu*ment defines which operation is to be performed.

cmd Flag	Use
F_SETLK	Sets a file lock. Do not block if this cannot succeed immediately
F_SETLKW	Sets a file lock and blocks the calling process until the lock is acquired
F_GETLK	Queries as to which process locked a specified region of a file

For file locking, the third argument to *fcntl is* an address of a *struct flock-typed* variable. This variable specifies a region of a file where the lock is to be set, unset, or queried. The *struct flock* is declared in the <fcntl.h> as:

struct flock
{
      short l_type;   // what lock to be set or to unlock file
      short l_whence;   // a reference address for the next field
      off_t l_start; //offset from the l_whence reference address
      off_t l_len;   // how many bytes in the locked region
      pid_t l_pid;   //PID of a process which has locked the file
};

The possible values of l_type are:

l_ type value	Use
F_RDLCK	Sets a a read (shared) lock on a specified region
F_WRLCK	Sets a write (exclusive) lock on a specified region
F_UNLCK	Unlocks a specified region

The possible values of l_whence and their uses are:

l_whence value	Use
SEEK_CUR	The *l_start* value is added to the current file pointer address.
SEEK_CUR	The *!_start* value is added to the current file pointer **Use** address
SEEK_SET	The *l_start* value is added to byte 0 of the file
SEEK_END	The *l_start* value ts'added to the end (current size) of the file