# ■ File Locking: Complete Explanation

1. What is File Locking?

File locking is a mechanism that restricts access to a file by multiple processes or threads to prevent conflicts such as:

- Two processes writing to the same file at the same time (data corruption).

- One process reading incomplete/incorrect data while another process is writing.

Locks enforce synchronization by controlling who can read/write at a given time.

2. Why File Locking is Needed

Without locking:

- Race Conditions: Two programs write to a log file at the same time → scrambled data.

- Partial Reads: A program reads a file while another is modifying it → inconsistent state.

- Database Safety: Databases use fine-grained file locks to avoid corruption.

3. Types of File Locks

There are several ways to classify file locks:

## (A) Based on Access Mode

1. Shared Lock (Read Lock):

- Multiple processes can hold a shared lock at the same time.

- They can all read the file.

- No one can write until all shared locks are released.

Example: Multiple users reading a configuration file.

2. Exclusive Lock (Write Lock):

- Only one process can hold an exclusive lock at a time.

- Prevents both reads and writes by others.

- Ensures atomicity of modifications.

Example: Updating a bank balance file.

## (B) Based on Duration

1. Advisory Lock:

- Processes must cooperate voluntarily to respect the lock.

- OS does not force it; if a process ignores the lock, it can still read/write.

Example: POSIX flock() or fcntl() advisory locks on Linux.

2. Mandatory Lock:

- Enforced by the OS kernel.

- If a process tries to read/write a locked file, the OS blocks it until the lock is released.

- Rare in practice (used in Windows and special Linux setups).

Example: Critical sections in system files.

## (C) Based on Scope

1. Whole-File Locking:

- Lock applies to the entire file.

- Simple, but less flexible (blocks other readers/writers completely).

Example: Simple text editors.

2. Record Locking (Byte-Range Locking):

- Lock applies to a portion (range of bytes) of the file.

- Multiple processes can safely work on different sections simultaneously.

Example: Database systems (locking rows stored in a file).

## (D) Based on Behavior

1. Blocking Lock:

- If a process requests a lock that’s unavailable, it waits until it becomes free.

Example: fcntl() with blocking mode.

2. Non-blocking Lock:

- If the lock is unavailable, the call fails immediately instead of waiting.

Example: flock(fd, LOCK_EX | LOCK_NB) in Linux.

## (E) Based on System

1. POSIX Locks (Unix/Linux):

- Advisory by default.

- Provided by flock(), fcntl(), or lockf().

2. Windows Locks:

- Usually mandatory.

- Provided by LockFile() and LockFileEx() APIs.

3. Distributed Locks:

- Used when files are shared across multiple machines.

- Examples:

- NFS (Network File System): Uses NLM (Network Lock Manager).

- Database Systems: Implement their own distributed locks.

- Cloud Systems: Redis/Zookeeper-based distributed locking.

4. How File Locking Works Internally

- The OS maintains a lock table (per process and per file descriptor).

- When a process requests a lock:

- The kernel checks if it conflicts with existing locks.

- If no conflict → lock granted.

- If conflict:

- For blocking locks → process sleeps until it’s free.

- For non-blocking locks → system call fails.

5. File Locking in Practice

Linux Examples

```
// Shared (read) lock using flock

flock(fd, LOCK_SH);

// Exclusive (write) lock

flock(fd, LOCK_EX);

// Non-blocking lock

flock(fd, LOCK_EX | LOCK_NB);

// Record locking with fcntl (locks 100 bytes starting at offset 200)

struct flock fl;

fl.l_type = F_WRLCK; // write lock

fl.l_whence = SEEK_SET;

fl.l_start = 200;

fl.l_len = 100;

fcntl(fd, F_SETLK, &fl;);
```

Windows Example

```
LockFile(hFile, 0, 0, MAXDWORD, MAXDWORD); // lock entire file

UnlockFile(hFile, 0, 0, MAXDWORD, MAXDWORD); // unlock
```

6. Problems with File Locking

1. Deadlocks:

- Two processes wait for each other’s lock indefinitely.

- Example: Process A locks part 1, Process B locks part 2 → both wait for the other.

2. Starvation:

- A process never gets a lock because others always grab it first.

3. Performance Overhead:

- Too much locking slows down throughput.

4. Distributed Locking Complexity:

- Maintaining locks across machines is hard (clock skew, network partitions).

7. Best Practices

- Use record locking for fine-grained control.

- Prefer advisory locks unless mandatory is absolutely needed.

- Always implement deadlock prevention (timeouts, lock ordering).

- For distributed systems, use a reliable lock manager (Zookeeper, etcd, Redis Redlock).

- Release locks as soon as possible.

■ Summary Table:

| Type | Subtype | Description |

|------------------|----------------------|------------------------------------------|

| Access Mode | Shared / Exclusive | Controls read vs. write access |

| Duration | Advisory / Mandatory | Voluntary vs. OS-enforced |

| Scope | Whole-file / Record | Entire file vs. portion of file |

| Behavior | Blocking / Non-blocking | Waits vs. fails immediately |

| System | POSIX / Windows / Distributed | OS and network-level locks |

| **Type** | **Subtype** | **Description** |
|---|---|---|
| Access Mode | Shared / Exclusive | Controls read vs. write access |
| Duration | Advisory / Mandatory | Voluntary vs. OS-enforced |
| Scope | Whole-file / Record | Entire file vs. portion of file |
| Behavior | Blocking / Non-blocking | Waits vs. fails immediately |
| System | POSIX / Windows / Distributed | OS and network-level locks |