System and Device Programming

File Locking

Stefano Quer

Dipartimento di Automatica e Informatica

Politecnico di Torino

File locking

- When two processes edit the same file at the same time the final state of the file corresponds to the last process that wrote the file
- However, several applications need to be certain that are the only one to write to a file
  - UNIX systems provide file locking
    - It is the term normally used to describe the ability of a process to prevent other processes from modifying a region of a file
  - File locking is a limited form of **synchronization**

File locking

- File locking is performed as a byte-range locking
  - A range of a file (possibly, the entire file) is locked
- For locking we use function fcntl
  - The parameter **cmd** must be set to
    - F_GETLK, F_SETLK, or F_SETLKW
  - The third argument must point to an **flock structure**

The commands

```
int fcntl (int fd, int cmd, struct flock *flockptr);
```

cmd	Purpose
F_GETFL or F_SETFL	Get/set file status flags
F_DUPFD or F_DUPFD_CLOEXEC	Duplicate an existing descriptor
F_GETFD or F_SETFD	Get/set file descriptor flags
F_GETOWN or F_SETOWN	Get/set asynchronous I/O ownership
F_GETLK, F_SETLK, or F_SETLKW	Get/set record locks

Three commands can be issued

The commands

- F_GETLK
  - **Check** whether the lock described by **flockptr** is blocked by some other lock
    - If a lock exists that would prevent ours from being created, the information on that existing lock overwrites the information pointed to by flockptr
    - If no lock exists that would prevent ours from being created, the structure pointed to by flockptr is left unchanged except for the l_type member, which is set to F_UNLCK

```
int fcntl (int fd, int cmd, struct flock *flockptr);
```

The commands

- F_SETLK
  - **Set the lock** described by **flockptr**
  - If the compatibility rule prevents the system from giving us the lock fcntl returns immediately with errno set to either EACCES or EAGAIN
  - This command is also used to clear the lock described by flockptr (l_type of F_UNLCK)

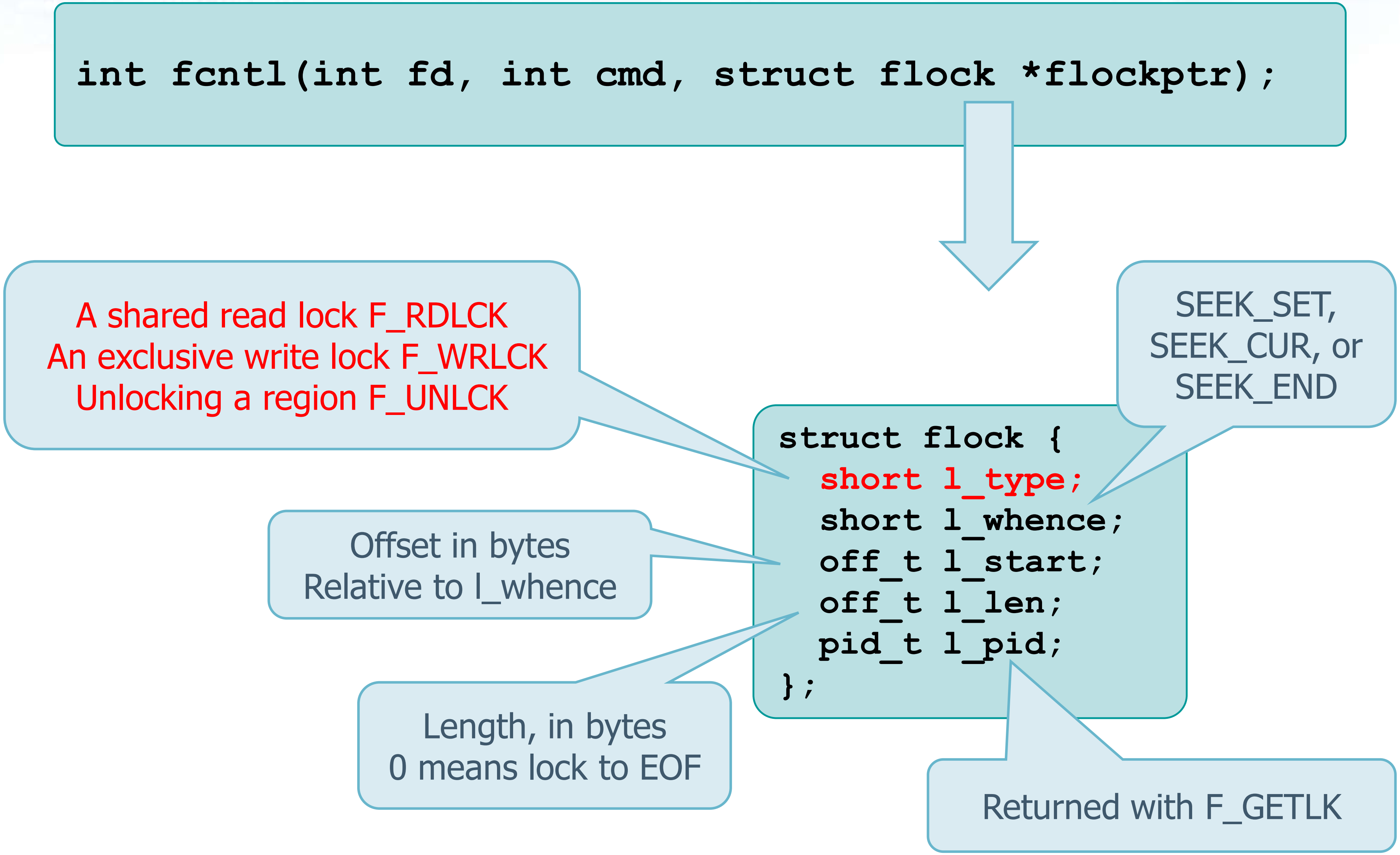
```
int fcntl (int fd, int cmd, struct flock *flockptr);
```

The commands

- F_SETLKW
  - **Set the lock** described by **flockptr** with a blocking operation
    - A blocking version of F_SETLK
    - The W in the command name means wait
  - If the requested read lock or write lock cannot be granted because another process currently has some part of the requested region locked, the calling process is put to sleep
  - The process wakes up either when the lock becomes available or when interrupted by a signal

```
int fcntl (int fd, int cmd, struct flock *flockptr);
```

The flock structure



```
int fcntl(int fd, int cmd, struct flock *flockptr);
```

A shared read lock F_RDLCK
An exclusive write lock F_WRLCK
Unlocking a region F_UNLCK

SEEK_SET,
SEEK_CUR, or
SEEK_END

```
struct flock {
  short l_type;
  short l_whence;
  off_t l_start;
  off_t l_len;
  pid_t l_pid;
};
```

Offset in bytes
Relative to l_whence

Length, in bytes
0 means lock to EOF

Returned with F_GETLK

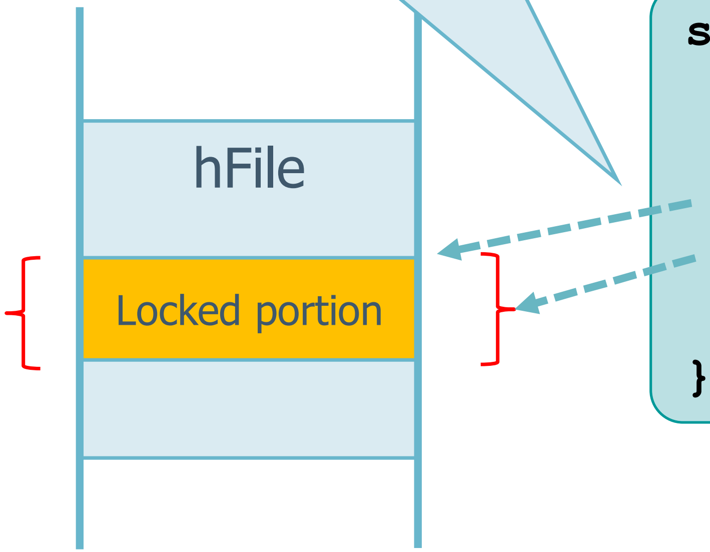
The flock structure

```
int fcntl(int fd, int cmd, struct flock *flockptr);
```

The starting byte offset of the region being locked or unlocked (l_start and l_whence)

SEEK_SET, SEEK_CUR, or SEEK_END

```
struct flock {
  short l_type;
  short l_whence;
  off_t l_start;
  off_t l_len;
  pid_t l_pid;
};
```



The size of the region in bytes (l_len)

Returned with command F_GETLK

The flock structure

```
int fcntl(int fd, int cmd, struct flock *flockptr);
```

```
struct flock {
  short l_type;
  short l_whence;
  off_t l_start;
  off_t l_len;
  pid_t l_pid;
};
```

SEEK_SET, SEEK_CUR, or SEEK_END

The ID (l_pid) of the process holding the lock that can block the current process (returned by F_GETLK only)

Returned with command F_GETLK

Guidelines

- Several rules apply
  - The two parameters specifying the starting offset of the region are similar to the last two arguments of the **lseek** function

```
off_t lseek (int fd, off_t offset, int whence);
```

  - Locks can start and extend beyond the current end of file, but cannot start or extend before the beginning of the file

```
struct flock {
  short l_type;
  short l_whence;
  off_t l_start;
  off_t l_len;
  pid_t l_pid;
};
```

Guidelines

- If **l_len** is 0, the lock extends to the largest possible offset of the file
  - This allows us to lock a region starting anywhere in the file, up through and including any data that is appended to the file
- To lock the entire file, we set **l_start** and **l_whence** to point to the beginning of the file and specify a length (**l_len**) of 0

```
struct flock {
  short l_type;
  short l_whence;
  off_t l_start;
  off_t l_len;
  pid_t l_pid;
};
```

Guidelines

- Repeated Lock Request
  - If a lock is present
  - When a new lock request is granted or refused ?

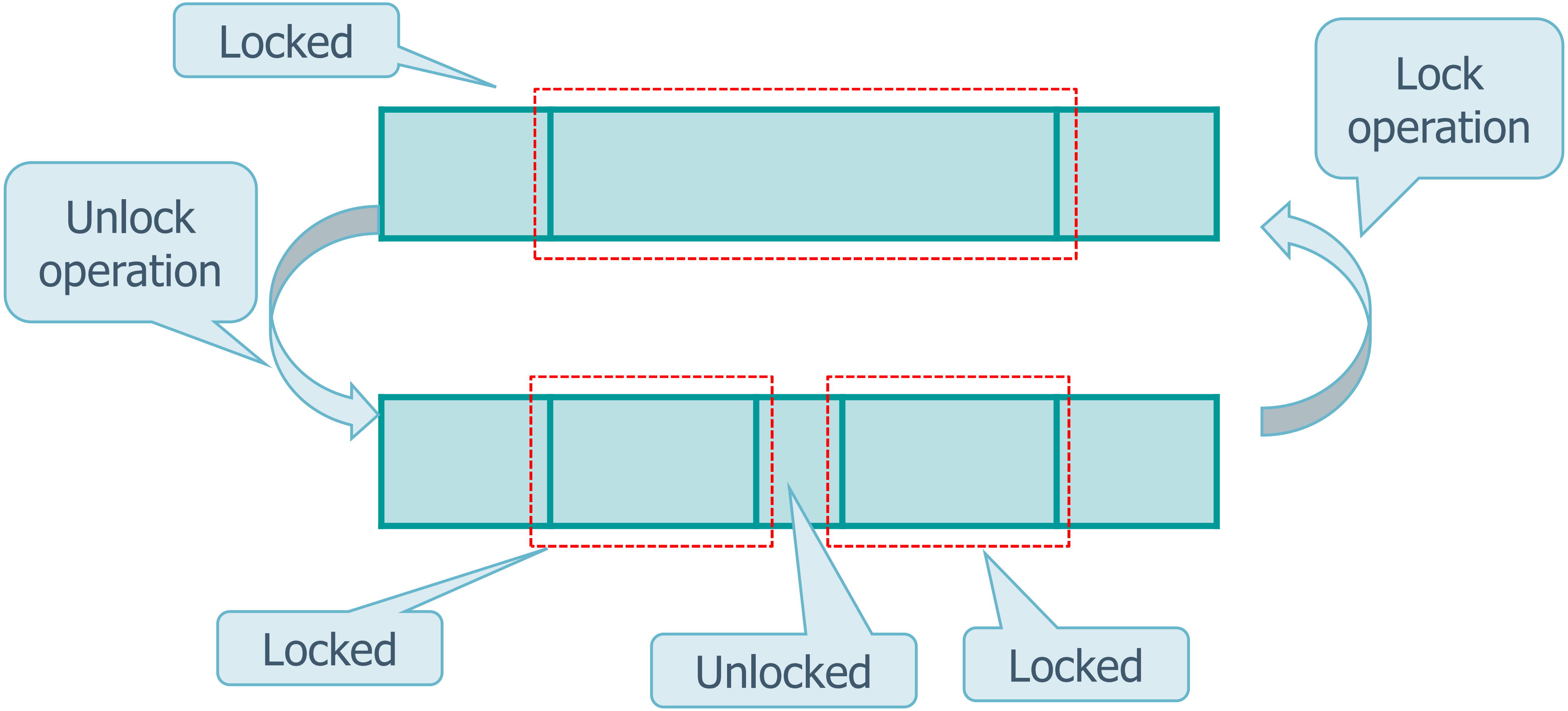
Exiting Lock	**Requested Lock Type**	
	Read Lock	**Write Lock**
None	Granted	Granted
Read lock	Granted	Refused
Write lock	Refused	Refused

Guidelines

- Lock **belongs** to a **process**, and it is possible to
  - The compatibility rule applies to lock requests made from different processes, not to multiple lock requests made by a single process
  - If a process has an existing lock on a range of a file, a subsequent attempt to place a lock on the same range by the same process will replace the existing lock with the new one
    - If a process has a write lock on bytes 16–32 of a file and then tries to place a read lock on bytes 16–32, the request will succeed, and the write lock will be replaced by a read lock

Guidelines

- When setting or releasing a lock on a file, the system combines or splits adjacent areas as required



Guidelines

- File locking can produce
  - Starvation
    - Processes A and B periodically obtain a shared lock whereas C is waiting forever for an exclusive lock
  - Deadlock
    - Process A is waiting for B to unlock and vice-versa (even on slightly a different file region)

Example

Lock a file region

```
#include <fcntl.h>

...

int lock_region (int fd, int cmd, int type, off_t offset,
  int whence, off_t len) {


  struct flock lock;

  lock.l_type = type;
  lock.l_start = offset;
  lock.l_whence = whence;
  lock.l_len = len;

  return (fcntl(fd, cmd, &lock));
}
```

Offset in bytes, relative to l_whence

A shared read lock F_RDLCK
An exclusive write lock F_WRLCK
Unlocking a reagionF_UNLCK

SEEK_SET, SEEK_CUR, or SEEK_END

Length, in bytes
0 means lock to EOF

-1 on error