FS_FILELOCKS

Purpose

Locks and/or unlocks a range (record) in a opened file.

Calling Sequence

```
int far pascal FS_FILELOCKS(psffsi, psffsd, pUnLockRange, pLockRange,
timeout,
                            flags)

struct sffsi far * psffsi;
struct sffsd far * psffsd;
struct filelock far * pUnLockRange;
struct filelock far * pLockRange;
unsigned long timeout;
unsigned long flags;
```

Where

psffsi is a pointer to the file-system-independent portion of an open file instance.

psffsd is a pointer to the file-system-dependent portion of an open file instance.

pUnLockRange is a pointer to a filelock structure, identifying the range of the file to be unlocked. The filelock structure has the following format:

```
struct filelock {
    unsigned long FileOffset;   /* offset where the lock/unlock begins  */
    unsigned long RangeLength;  /* length of region locked/unlocked     */
}
```

If *RangeLength* is zero, no unlocking is required.

pLockRange is a pointer to a filelock structure, identifying the range of the file to be locked. If *RangeLength* is zero, no locking is required.

timeout is the maximum time in milliseconds that the requester wants to wait for the requested ranges, if they are not immediately available.

flags is the bit mask which specifies what actions are to taken:

SHARE Bit 0 on	indicates other processes can share access to this locked range. Ranges with SHARE bit on can overlap.
SHARE Bit 0 off	indicates the current process has exclusive access to the locked range. A range with the SHARE bit off CANNOT overlap with any other lock range.
ATOMIC Bit 1 on	indicates an atomic lock request. If the lock range equals the unlock range, an atomic lock will occur. If the ranges are not equal, an error will be returned.
All other bits (2-31) are reserved and must be zero.	

Remarks

This entry point was added to support the 32-bit **DosSetFileLocks** API.

If the lock and unlock range lengths are both zero, an error, *ERROR_LOCK_VIOLATION* will be returned to the caller. If only a lock is desired, *pUnLockRange* can be NULL or both *FileOffset* and *RangeLength* should be set to zero when the call is made. The opposite is true for an unlock.

When the atomic bit is not set, the unlock occurs first then the lock is performed. If an error occurs on the unlock, an error is returned and the lock is not performed. If an error occurs on the lock, an error is returned and the unlock remains in effect if one was requested. If the atomic bit is set and the unlock range equals the lock range and the unlock range has shared access but wants to change the access to exclusive access, the function is atomic. FSDs may not support atomic lock functions. If error *ERROR_ATOMIC_LOCK_NOT_SUPPORTED* is returned, the application should do an unlock and lock the range using nonatomic operations. The application should also be sure to refresh its internal buffers prior to making any modifications.

Closing a file with locks still in force causes the locks to be released in no defined order.

Terminating a process with a file open and having issued locks on that file causes the file to be closed and the locks to be released in no defined order.

The figure below describes the level of access granted when the accessed region is locked. The locked regions can be anywhere in the logical file. Locking beyond end-of-file is not an error. It is expected that the time in which regions are locked will be short. Duplicating the handle duplicates access to the locked regions. Access to the locked regions is not duplicated across the *DosExecPgm* system call. The proper method for using locks is not to rely on being denied read or write access, but attempting to lock the region desired and examining the error code.

Locked Access Table

Action	**Exclusive Lock**	**Shared Lock**
Owner read	Success	Success
Non-owner read	Return code, not block	Success
Owner write	Success	Return code, not block
Non-owner write	Return code, not block	Return code, not block

The locked access table has the actions on the left as to whether owners or non-owners of a file do either reads or writes of files that have exclusive or shared locks set. A range to be locked for exclusive access must first be cleared of any locked subranges or locked any locked subranges or locked overlapping ranges.