



BAHIRDAR UNIVERSITY

FACULTY OF COMPUTING

SOFTWARE ENGINEERING DEPARTEMENT

Operating system and system programing second assignment

Name:-Yikeber Misganaw

ID:-1308331

Submitted to :- lecturer wondimu

Submission date 17 -11 -2014 EC.

Flock() system call

What is system call?

A **system call** is a mechanism that provides the interface between a process and the operating system. It is a programmatic method in which a computer program requests a service from the kernel of the OS.

System call offers the services of the operating system to the user programs via API (Application Programming Interface). System calls are the only entry points for the kernel system.

What is flock() system call?

File locking is a mechanism to restrict access to a file among multiple processes. It allows only one process to access the file in a specific time, thus avoiding the interceding update problem. A call to `flock()` may block if an incompatible lock is held by another process. To make a nonblocking request, include **LOCK_NB** (by ORing) with any of the above operations.

A single file may not simultaneously have both shared and exclusive locks. Locks created by **flock()** are associated with an open file table entry. This means that duplicate file descriptors (created by, for example, *fork(2)* or *dup(2)*) refer to the same lock, and this lock may be modified or released using any of these descriptors. Furthermore, the lock is released either by an explicit **LOCK_UN** operation on any of these duplicate descriptors, or when all such descriptors have been closed. If a process uses *open(2)* (or similar) to obtain more than one descriptor for the same file, these descriptors are treated independently by **flock()**. An attempt to lock the file using one of these file descriptors may be denied by a lock that the calling process has already placed via another descriptor.

A process may only hold one type of lock (shared or exclusive) on a file. Subsequent **flock()** calls on an already locked file will convert an existing lock to the new lock model.

Apply or remove an advisory lock on the open file specified by *fd*.

List of parameters

In `int flock(int fd , int operation)` there are two parameters:-

✓ fd and operation

The parameter *operation* is one of the following:

A call to **flock()** may block if an incompatible lock is held by another process. To make a non-blocking request, include **LOCK_NB** (by *OArg*) with any of the above operations.

A single file may not simultaneously have both shared and exclusive locks.

Locks created by **flock()** are associated with an open file table entry. This means that duplicate file descriptors (created by, for example, **fork(2)** or **dup(2)**) refer to the same lock, and this lock may be modified or released using any of these descriptors. Furthermore, the lock is released either by an explicit **LOCK_UN** operation on any of these duplicate descriptors, or when all such descriptors have been closed.

If a process uses **open(2)** (or similar) to obtain more than one descriptor for the same file, these descriptors are treated independently by **flock()**. An attempt to lock the file using one of these file descriptors may be denied by a lock that the calling process has already placed via another descriptor.

A process may only hold one type of lock (shared or exclusive) on a file. Subsequent **flock()** calls on an already locked file will convert an existing lock to the new lock mode.

Locks created by **flock()** are preserved across an **execve(2)**.

A shared or exclusive lock can be placed on a file regardless of the mode in which the file was opened.

Arguments:

filedes

The file descriptor of an open file.

operation

What you want to do to the file; one of the following:

- **LOCK_EX**—apply an exclusive lock.
- **LOCK_SH**—apply a shared lock.
- **LOCK_UN**—unlock an existing lock.

You can OR **LOCK_NB** with **LOCK_EX** or **LOCK_SH** to prevent the function from blocking (see below).

Flock () or file locking system call

Library:

libc

Use the `-l c` option to [gcc](#) to link against this library. This library is usually included automatically.

Description:

The *flock()* function applies or removes an advisory lock on the file associated with the open file descriptor *filedes*.

Advisory locks allow cooperating processes to perform consistent operations on files, but they don't guarantee consistency.

The locking mechanism allows two types of locks: *shared* and *exclusive*. At any time, multiple shared locks may be applied to a file, but at no time are multiple exclusive, or both shared and exclusive, locks allowed simultaneously on a file.

A shared lock may be upgraded to an exclusive lock, and vice versa, by specifying the appropriate lock type. The previous lock is released and a new lock is applied (possibly after other processes have gained and released the lock).

Requesting a lock on an object that's already locked causes the caller to be blocked until the lock may be acquired. If you don't want the caller to be blocked, you can specify `LOCK_NB` in the operation to make the call fail instead (*errno* is set to `EWOULDBLOCK`).

Locks are applied to files, not file descriptors. That is, file descriptors duplicated through [dup\(\)](#) or [fork\(\)](#) don't result in multiple instances of a lock, but rather multiple references to a single lock. If a process holding a lock on a file forks, and the child explicitly unlocks the file, the parent loses its lock.

In QNX Neutrino 7.0 or later, if you open the same file multiple times in a process, an *flock()* on one file descriptor referring to a file locks out an *flock()* on a different fd.

Processes blocked awaiting a lock may be awakened by signals.

Returns:

0

The operation was successful.

-1

An error occurred (*errno* is set).

Errors:

EBADF

Flock () or file locking system call

Invalid descriptor, *filedes*. *fd is not a not an open file descriptor*

EINVAL

The argument operation doesn't include one of LOCK_EX, LOCK_SH, or LOCK_UN.

ENOMEM

The system can't allocate sufficient memory to store lock resources.

ENOSYS

The filesystem doesn't support the operation.

EOPNOTSUPP

The *filedes* argument refers to an object other than a file.

EWOULDBLOCK

The file is already locked, and LOCK_NB was specified.

Usage of the flock function in C language in Linux

Header file # include <sys/file. h>

Defines the int flock (int fd, int operation) function ;

Function Description: Flock () performs various locking or unlocking actions on the file referred to by the parameter FD according to the method specified by the parameter operation. This function can only lock the entire file and cannot lock a certain area of the file.

Operation has the following four conditions:

Lock_sh creates a shared lock. Multiple processes can share and lock the same file at the same time.

Lock_ex creates a mutex lock. One file has only one mutex lock.

Lock_un unlocks the file.

Lock_nb cannot be used to establish a lock. This operation is not blocked and the process is returned immediately. Usually it is combined with lock_sh or lock_ex or (!.

Shared locks and mutex locks cannot be created for a single file. When DUP () or fork () is used, the file description does not inherit the lock.

If the return value is 0, a success is returned. If an error exists,-1 is returned. The error code is stored in errno.

Flock only needs to read and write the file after opening the file, and then flock it after use, lock

Flock () or file locking system call

the front and unlock the back. It is actually so simple, but some problems were found some time ago. The problem is described as follows:

One process opens the file, input an integer, and then the last write lock (lock_ex), then enter an integer to unlock (lock_un), and the other process opens the same file, write data directly to the file and find that the lock does not work and can be written normally (I am using a Super User). Google found that flock does not provide a lock check, that is to say, before using flock, the user needs to check whether the lock has been applied, it indicates that the white point is to use flock to check whether the file is locked. If the flock is locked, it will block it (an attempt Lock the file using one of these file descriptors may be denied by a lock that the calling process has already placed via another descriptor) Unless lock_nb is used. A complete example code for testing is as follows:

```
// Lockfile. c

//testprocess.c

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <errno.h>

#include <sys/file.h>

int main()

{

    int fd,i;

    char path[]="/home/taoyong/test.txt";

    char s[]="writing.../nwriting...../n";

    extern int errno;

    fd=open(path,O_WRONLY|O_CREAT|O_APPEND);

    if(fd!=-1)

    {

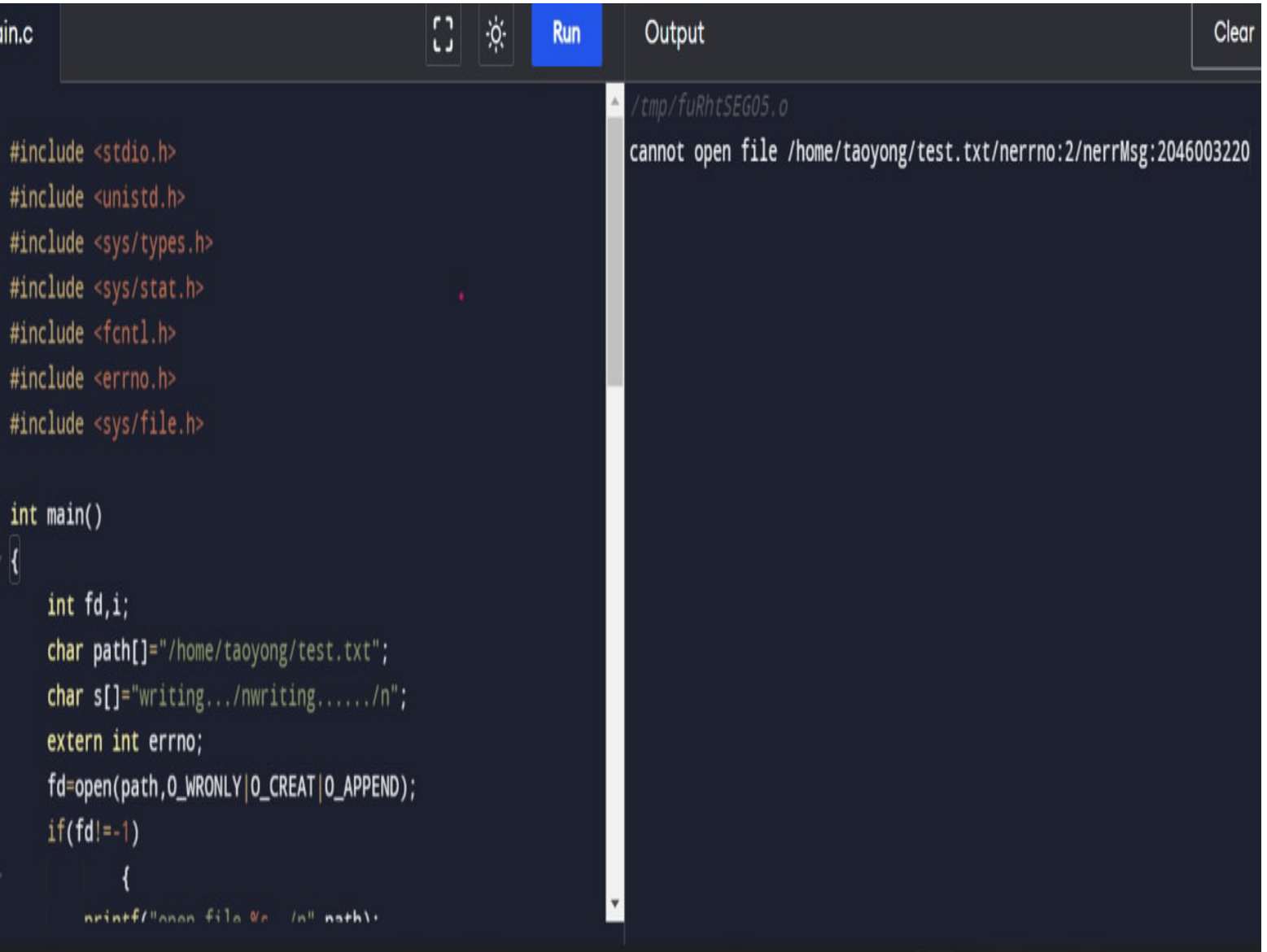
        printf("open file %s ./n",path);
```

```
if(flock(fd,LOCK_EX|LOCK_NB)==0)
{
printf("the file was locked by the process./n");
if(-1!=write(fd,s,sizeof(s)))
{
printf("write %s to the file %s/n",s,path);
}
else
{
printf("cannot write the file %s/n",path);
printf("errno:%d/n",errno);
printf("errMsg:%d/n",strerror(errno));
}
}
else
{
printf("the file was locked by other process.Can't write.../n");
printf("errno:%d:",errno);
}

close(fd);
}
else
{
printf("cannot open file %s/n",path);
```

Flock () or file locking system call

```
printf("errno:%d/n",errno);  
printf("errMsg:%d",strerror(errno));  
}  
}
```



The screenshot shows a code editor with a dark theme. On the left, a C program is visible, including headers like `<stdio.h>`, `<unistd.h>`, `<sys/types.h>`, `<sys/stat.h>`, `<fcntl.h>`, `<errno.h>`, and `<sys/file.h>`. The `main` function starts with variable declarations for `fd` and `i`, and a path string `path="/home/taoyong/test.txt"`. It attempts to open the file with `fd=open(path,O_WRONLY|O_CREAT|O_APPEND);` and checks if `fd != -1`. The right pane shows the output of the program, which is an error message: `cannot open file /home/taoyong/test.txt/nerrno:2/nerrMsg:2046003220`. The output pane also shows the temporary file path `/tmp/fuRhTSEG05.o` at the top.

```
#include <stdio.h>  
#include <unistd.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <errno.h>  
#include <sys/file.h>  
  
int main()  
{  
    int fd,i;  
    char path[]="/home/taoyong/test.txt";  
    char s[]="writing.../nwriting...../n";  
    extern int errno;  
    fd=open(path,O_WRONLY|O_CREAT|O_APPEND);  
    if(fd!=-1)  
    {  
        printf("open file %s /n" path);  
    }  
}
```

Output: /tmp/fuRhTSEG05.o
cannot open file /home/taoyong/test.txt/nerrno:2/nerrMsg:2046003220

The out put of the above code is:This program works for locking a file if there is in the file directory

Since there is no file to be locked on `/home/taoyong/test.txt` path it display the else stathe abobe message is displayed

Summary

A call to flock() may block if an incompatible lock is held by another process. To make a nonblocking request, include LOCK_NB (by ORing) with any of the above operations. A single file may not simultaneously have both shared and exclusive locks. Locks created by flock() are associated with an open file description (This means that duplicate file descriptors (created by, for example, fork(2) or dup(2)) refer to the same lock, and this lock may be modified or released using any of these file descriptors. Furthermore, the lock is released either by an explicit LOCK_UN operation on any of these duplication file descriptors, or when all such file descriptors have been closed. If a process uses open(2) (or similar) to obtain more than one file descriptor for the same file, these file descriptors are treated independently by flock(). An attempt to lock the file using one of these file descriptors may be denied by a lock that the calling process has already placed via another file descriptor.