ECE391 Computer System Engineering Lecture 26

Dr. Zbigniew Kalbarczyk
University of Illinois at Urbana- Champaign

Fall 2018

Lecture Topics

- Driver design process
- I-mail design

Aministrivia

- MP3.5 Code cutoff:
 - 5:00pm Monday, December 10
- Hand-in/demo
 - 0.5 h (all members should be there)
 - (Monday night); Tuesday; Wednesday
- Post time slots the next week
 - First come first served

- Competition
 - Thursday December 13, in the ECE391 Lab

Aministrivia

- Final Exam:
 - 8:00am to 11:00am, Friday, December 14

- Review session
 - During the Lecture Time om Thursday, December 6

I-mail and Device Driver Design

- Example is Illinois mail (I-mail)
 - instant messaging within the Linux kernel
 - (perhaps not a best idea) but good example
 - somewhat complex abstraction
 - issues similar to other communication drivers (e.g., TCP/IP)
 - simpler than most real implementations
 - can be described from ground up

Course Notes 6a on the class web site

Driver Design Process

- "contemplate" security (hard/impossible to add in correctly as afterthought!)
- 1. write out all ops (in terms of visible interface)
- 2. design data structures
- 3. pick a locking strategy
- 4. determine locks needed, pick lock ordering
- 5. consider blocking issues & wakeup

Driver Design Process

- 6. consider dynamic allocation issues & hazards
- 7. write code
- 8. write subfunctions and synchronization rules
- 9. return to 3 or 4 if 7 or 8 fails
- 10. unit test

I-mail Design - Security

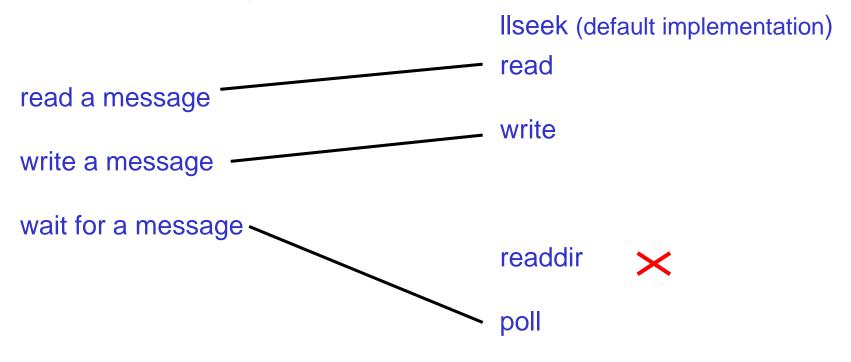
- Could just use the sysadmin for admin functions
- Instead, have I-mail administrator as a user with privileges
 - allows sysadmin to hand off I-mail rights
 - without giving away other rights

File Operations Structure include/linux/fs.h

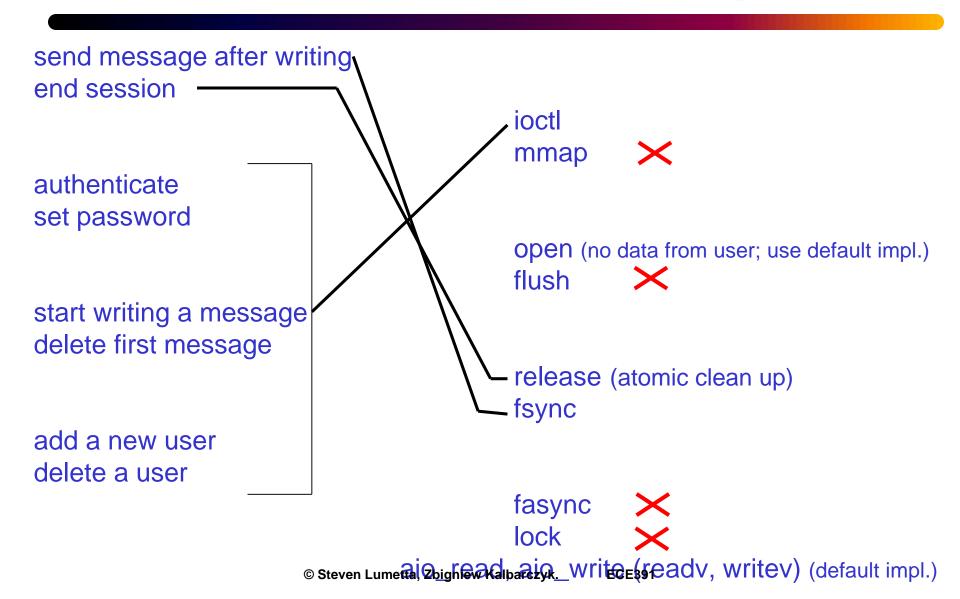
```
struct file operations {
        struct module *owner;
        loff t (*llseek) (struct file *, loff t, int);
        ssize t (*read) (struct file *, char user *, size t, loff t *);
        ssize t (*write) (struct file *, const char user *, size t, loff t *);
        ssize t (*aio read) (struct kiocb *, const struct iovec *, unsigned long, loff t);
        ssize t (*aio write) (struct kiocb *, const struct iovec *, unsigned long, loff t);
        int (*readdir) (struct file *, void *, filldir t);
        unsigned int (*poll) (struct file *, struct poll table struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        long (*unlocked ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm area struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl owner t id);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *, int datasync);
        int (*aio fsync) (struct kiocb *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file lock *);
        /* ... plus a couple more rarely used functions */
};
                           © Steven Lumetta, Zbigniew Kalbarczyk.
                                                           ECE391
```

I-mail Design - Operations

 I-mail operations (and the system calls we'll use to execute them)



I-mail Design - Operations



I-mail Design - Operations by Entry Point

- Abstractly, a user has
 - authentication data
 - an association with a program (one file at a time)
 - a list of messages (reads only the first)
 - possibly a message being written

I-mail Design - Operations by Entry Point

- read
 - get data from first message in user's list
 - block until message arrives if necessary

- write
 - write data into message being written

I-mail Design - Operations by Entry Point

poll

- check status of message list (empty/not empty)
- block until message arrives if necessary

release

- send the message being written
- remove file association from user data

fsync

send the message being written

I-mail Design - ioctl Operations

- authenticate
 - associate file with user data

- set password
 - change authentication data
- start write message
 - send the message being written
 - start writing a new message

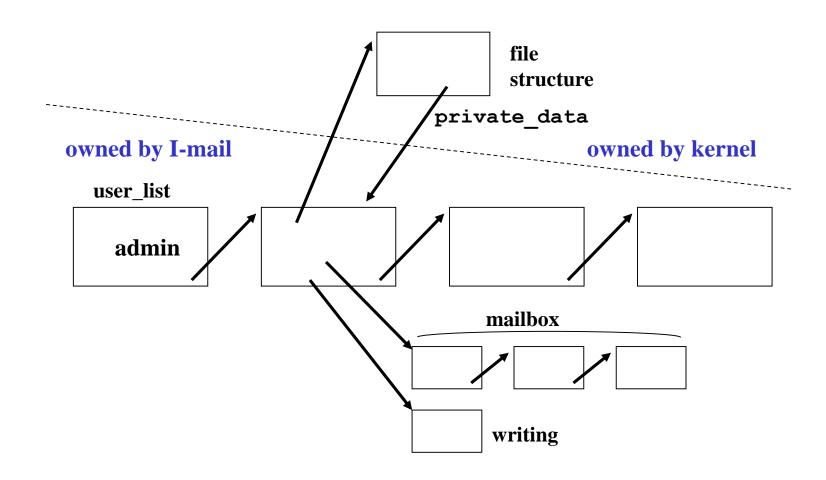
I-mail Design - ioctl Operations

- delete message
 - remove first message in list
 - reset file position
- add new user
 - check for a name conflict
 - create the new user
- delete user
 - kick off active user
 - delete messages and user

I-mail – Data Structures

- We want a user-level admin rather than requiring machine sysadmin privileges
- admin can't be deleted
- make it the head of the user list

I-mail – Data Structures



I-mail – locks

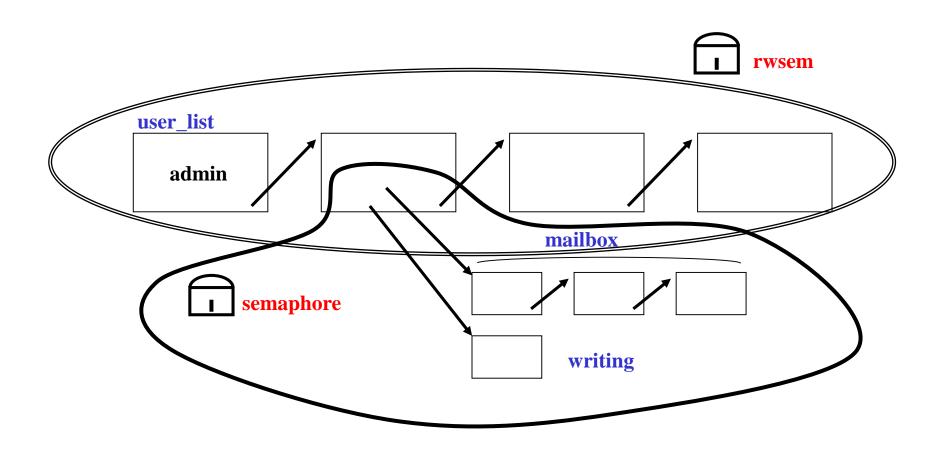
For the user list

- lots of reads, few writes
- want concurrent authorization and management
- use a reader/writer semaphore (rwsem)

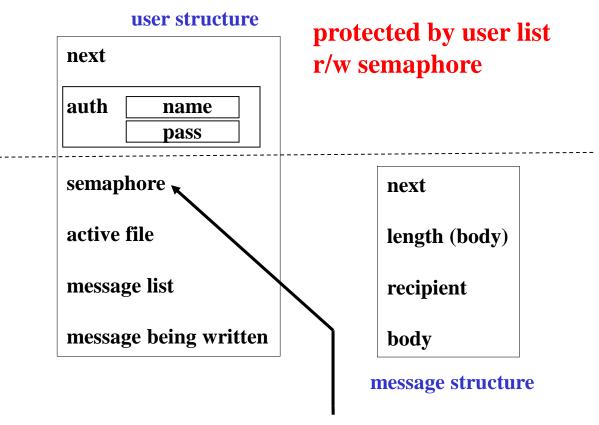
For the users

- lots of reads and writes
- typically by one user process
- use a semaphore

I-mail – locks



I-mail – Data Structures with Lock Annotations



protected by user data semaphore

I-mail – Lock Order

- I-mail lock order
 - 1. user list r/w semaphore
 - 2. user data semaphore

- Q: Can you acquire >1 user semaphore simultaneously?
 - A: Absolutely not!
- Example: When delivering a message, at some point will need access to both user data structures; simply cannot hold them at same time.

Exercise: What locks are necessary for each op?

- read
 - user data sem.
- write
 - user data sem.
- poll
 - user data sem.
- fsync
 - user data sem.
 - What about delivery?
 - read user list r/w sem.
 - recipient's user data sem.
- release
 - user data sem. (and send message as above)

Exercise: What locks are necessary for each op?

(ioctl's)

- authenticate
 - read user list
 - user data sem. (if matched to a user)
- set password
 - write user list
 - (no need for user data sem.)
- write message
 - (may need to send message)
 - read user list
 - sender's user data sem. (if recipient validated)
- delete message
 - user data sem.

Exercise: What locks are necessary for each op?

add new user

- write user list
- (no <u>need</u> for user data sem.;
 no one can authenticate while we have write lock on user list)

delete user

- write user list
- user data sem. (<u>MUST</u> synchronize with all ops somehow!)