Some answers for the closed section

Subproblem 1.b

`moe` can write a setuid program, *or* `moe` can just make a copy of `/bin/kill` and make it setuid `moe` and executable only by members of group `stooges`.

Problem 2

The key phrase found on page 201 of the text is "a process never executes in user mode before handling outstanding signals." So if a process begin executing in user mode with no outstanding signals, though possibly with its stack rigged to invoke some signal handlers, there is no way for a signal to be delivered to that process unless some transition is made to kernel mode, either by an interrupt or system call. Consequently, there's no point in checking for signals when making the transition from user to kernel mode, because there won't be any to check for.

Problem 3

initial test for the lock on the inode $\approx$ carrier sense

test of lock after sleep $\approx$ collision detect

Uh. You see, if two processes wake up at the same time one will notice the lock is already taken. Sort of. More or less. You get the idea.

Problem 4

page #	real
0	1900
1	47000
2	38000
3	17000
4	on disk

It would be ok to put 3000 instead of 3 in the left column.

Subproblem 5.a

You need to give your device a number, let's use α.

You need to write appropriate read, write, and open routines. Write is trivial. Just use `/dev/null`'s. Not clear open really needs to do anything. Read is more interesting. The device read routine must read the CSR, *i.e.,* it must read address 0174456. If bit 13 of the CSR is on, a /tt 'B' is placed in the read buffer. Otherwise, a /tt 'F' is put in the read buffer. In either case, the device read routine must indicate that only one byte was read.

Finally, the device I/O routines must be installed in position α of the device switch table.

Subproblem 5.b

```
# mknod /dev/etherstat c α 0
```

Subproblem 6.a

Changing the access permissions of `/nowrite/mtpnt` means changing a disk inode on a read-only file system and that is not allowed.

Subproblem 6.b

The inode mount point flag is part of the in-core inode and *not* part of the disk inode (p. 63, Back). Therefore a mount operation changes kernel memory, not disks, so the usual mount algorithm should work fine. (By the way, I never really tested this, but the logic seems reasonable.)

Subproblem 6.c

Now changing the access permissions of `/nowrite/mtpnt` means changing the disk inode of the root of the file system `/dev/dsk5`. That can be done as long as `/dev/dsk5` is mounted read/write.

Subproblem 7.d

There are several ways. For example, the parent `kill`s the child because it executes any statements.

Subproblem 7.e

The parent `kill`s the child after the child `write`s `'B'` to the pipe but before the child `read`s from the pipe.

Subproblem 7.f

A semi-tough one. Again, there is more than one correct answer.

The parent executes its first two statements, 11 and 12. It reads the `'A'` from the pipe. A context switch occurs.

The child executes. It writes a `'B'` to the pipe, reads the `'B'` back from the pipe, and writes the `'B'` to standard output. It then exits, and another context switch occurs.

The parent continues. It writes (statement 13) the `'A'` it had read before the context switch. The pipe now has no writers, so the parent's read of the pipe detects the end-of-file and does not change the value of character `b`. Thus, the `'A'` is written a second time.