Midterm #2

Some answers for the closed section

Problem 1.

The parent process creates a pipe and forks a child process. The parent process then writes `'y'` to the pipe and closes its writing end of the pipe. Then the parent and child both try to read a character from the shared pipe. It's a race.

If the parent wins, it will read the `'y'` back into variable `c` while the child is blocked on its `read` call. The parent then writes `'y'` to standard output, kills the child process by sending it a `SIGBUS` signal, and waits for the child to be terminated. Thus, if the parent *wins,* the string `"y"` is written to standard output.

If the child wins, it will read the `'y'` back into variable `c` while the parent is blocked on its `read` call. The child then forks a grandchild process. The value of `pid` is zero for both child and grandchild, so both set `c` to `'z'` and both then write `'z'` to standard output. After both child and grandchild exit, no process will have the write end of the pipe open. The parent will then return from the `read` call having detected the end-of-file condition on the pipe. It will write a `'y'` to standard output, send an ignored `SIGBUS` signal to its child, and then `wait` on its zombie offspring. Thus, if the child *wins* the race to read the pipe, the string `"zzy"` is written to standard output.

The common mistake was assuming the child would always read the character from the pipe rather than recognizing the race. (By the way, I ran this program more than 200 times, and the parent always won the race.) Another common mistake was assuming that the child's `fork` somehow changed the value of `pid.`

Problem 2

The program `suggest` must be owned by root, and its setuid bit must be on. However, `suggest` must be careful that the person executing it actually has read access to the input file. Otherwise, `smith` could type:

```
% suggest smith /unc/jones/secret.file
```

to copy `jones`' secret file to his own `suggestions` file. Also, `suggest` must be careful that `smith` actually has write access to his `suggestions` file. Think about what would happen if `smith` had previously executed the command:

```
% ln /etc/passwd /unc/smith/suggestions
```

Implementing `suggest` to avoid security problems requires either a careful use of the `stat` system call to check permissions or a careful use of `seteuid` to switch between the two user ID's. (The obvious use of `setuid` won't work because when `root` executes `setuid` all three user ID's are changed.)

The usual grade for this question was six. Most answers mentioned "setuid" but were pretty vague about how it is used.

Problem 3.

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.

Only about three people gave the correct answer. (Unfortunately, I wouldn't have been in that group had I been taking the test.) Because many of the "answers" were rephrasings of the question and because many people didn't give any answer at all and because these two cases didn't seem that much different to me, a lot of zeros were given for this question.

Distribution

range	number in range
90-93	2
80-89	4
70-79	15
60-69	8
50-59	2

Yes, it was too long a test, and most peoples grades are lower because of the time problem. Next week, we'll try to give every one a list of the grades we have recorded for him or her.