Final exam–May 2, 2:00 PM

Open book section (60 points)

The exam is to be turned in at 5:00 PM. The closed book section should be turned in before you open your books and notes to work the open book section. For the open book section, write your answers on separate pieces of paper.

Problem 1: (5 points)

Suppose a group called `stooges` has been created with users `moe`, `curley`, and `larry`.

Subproblem 1.a: (1 point)

Can `curley` create a file which is readable only by `stooges`? How?

Subproblem 1.b: (4 points)

Suppose `moe` wants to create a program `nyack` that allows either `curley` or `larry` to send signals to any process `moe` is running as if `moe` sent the signal himself. Can `moe` (assuming he has worked hard in Comp 190) write this program? Can he restrict `nyack` so that only `stooges` can run it?

Problem 2: (5 points)

A process checks for signals when it enters or leaves the sleep state (if it sleeps at an interruptible priority) and when it returns to user mode from the kernel after completion of a system call or after handling an interrupt. Why does the process not have to check for signals when entering the system for execution of a system call?

This problem is exercise 12 on page 241 of Bach's book.

Problem 3: (6 points)

A networking hacker, having just studied the Unix code for obtaining locked inodes (Figure 4.3, page 64 of Bach), proclaims: "It's just like CSMA/CD." Questioned further he mutters something about sleeping on a locked inode and then checking the lock when awakened.

What do think is on the mind of this displaced communications geek?

Problem 4: (5 points)

We are snooping on a process running on a machine where the page size is 1000 bytes. We notice the following virtual to read address mappings: virtual 115 to real 19115, virtual 1492 to real 47492, virtual 1989 to real 47989, virtual 2338 to real 38338, and virtual 3456 to real 17456. We then observe that accessing virtual address 4545 results in a page fault.

Draw a page table for the first five pages used by this process.

Problem 5: (10 points)

You have been requested to write a someone unusual (and perhaps silly) device driver that will be used to determine whether or not some machine is transmitting on the Ethernet. Your device driver should behave as follows:

(1) No matter how many bytes you try to read from this device, it will never actually read more than one byte. Thus "`read(fd, buff, 1000)`" and "`read(fd, buff, 1)`" both only change the first byte of `buff`.
(2) The read byte is `'B'` if the Ethernet is busy and `'F'` it is not.
(3) All writes to this device have no effect. They are all successful, but any written data is ignored. (Writes to this device are just like writes to `/dev/null`.)
(4) Your machine has a DEQNA ethernet interface like the one described in class and in Comer's book. The only facts you need to know about the DEQNA is that it's control and status register is stored at bus location 0174456 and that bit 13 of the CSR is 1 if a carrier is present on the ethernet and bit 12 of the CSR is 1 if the fuse is ok.

Subproblem 5.a: (7 points)

Outline with precision and brevity, how you will modify the kernel to implement this device driver.

Subproblem 5.b: (3 points)

Suppose the device driver is to be accessed via the special file `/dev/etherstat`. Describe how you must modify the file system to make this device accessible to user programs.

Problem 6: (11 points)

In Unix it is possible to mount a file system read-only by specifying the "`ro`" option to the "`mount`" command. Suppose a read-only file system `/dev/cd0` that has been mounted on directory `/nowrite` and that the root directory of this mounted file system contains a subdirectory `mtpnt`. Thus `/nowrite/mtpnt` is now directory within the read-only file system.

Subproblem 6.a: (2 points)

Attempts by the superuser to change the access permissions of `/nowrite/mtpnt` fail. Why?

Subproblem 6.b: (6 points)

Now, suppose the superuser wishes to mount a new file system `/dev/dsk5` on the directory `/nowrite/mtpnt`. The manual page for the `mount` system call does not seem to prohibit the use of a read-only directory as a mount point. That is, mounting on top of directories of read-only file systems is not listed as an error. How can the operating system perform a mount on a read-only file system?

Subproblem 6.c: (3 points)

Assuming the mount succeeds, now what happens if the superuser tries to change the access permissions of `/nowrite/mtpnt`?

Problem 7: (18 points)

The following program has been successfully compiled and run 100 times on a Unix system. It wrote "`AA`" to standard output 47 times, it wrote "`AB`" once, it wrote "`ABA`" 39 times, it wrote "`ABB`" 12 times, and it wrote "`BAA`" once.

Subproblem 7.a: (3 points)

Draw a picture showing the pipes created by the processes.

Subproblem 7.b: (2 points)

Why is two the least number of characters that can be written to standard output?

Subproblem 7.c: (2 points)

Why is three the greatest number of characters that can be written?

Subproblem 7.d: (3 points)

Describe how the program can produce "`AA`" as its output.

Subproblem 7.e: (4 points)

Describe how the program can produce "`AB`" as its output.

Subproblem 7.f: (4 points)

Describe how the program can produce "`BAA`" as its output.

```
#include <signal.h>
/*   1 */  main(argc, argv)
/*   2 */      int argc;
/*   3 */      char *argv[];
/*   4 */    {
/*   5 */      int p[2], pid;
/*   6 */      char b = 'X';
/*   7 */      pipe(p);
/*   8 */      write(p[1], "A", 1);
/*   9 */      pid = fork();
/*  10 */      if (pid > 0) {
/*  11 */          close(p[1]);
/*  12 */          read(p[0], &b, 1);
/*  13 */          write(1, &b, 1);
/*  14 */          kill(pid, SIGBUS);
/*  15 */          read(p[0], &b, 1);
/*  16 */          write(1, &b, 1);
/*  17 */          wait(0); }
/*  18 */      else {
/*  19 */          write(p[1], "B", 1);
/*  20 */          read(p[0], &b, 1);
/*  21 */          write(1, &b, 1);
/*  22 */          exit(0); }}
```