Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.828 Operating System Engineering: Fall 2003

Quiz II

All problems are open-ended questions. In order to receive credit you must answer the question as precisely as possible. You have 80 minutes to answer this quiz.

Write your name on this cover sheet AND at the bottom of each page of this booklet.

Some questions may be much harder than others. Read them all through first and attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES QUIZ.

1 (xx/15)	2 (xx/25)	3 (xx/20)	4 (xx/25)	5 (xx/15)	Total (xx/100)

Name:

I Control-C and the shell

1. [15 points]: In UNIX when a user is running a program from the shell, the user can terminate the program by typing ctrl-C. How would you change the 6.828 kernel and its shell to support this feature? Be careful, make sure when a user types the name of the program and hits ctrl-C before the program runs that the right thing happens (i.e., don't kill the shell itself or the file system). (Sketch an implementation and define "what the right thing" is.)

II CPU scheduling

2. [10 points]: `primespipe` spawns many environments until it has generated a prime greater than some specified number. If you modify your shell for the 6.828 kernel to run `primespipe` in the background and then type `ls` at your shell, it can happen that the output of `ls` won't show before `primespipe` completes. Explain why.

(Keep it brief)

3. [5 points]: Describe a solution to the problem described in the previous question.

(Keep it brief; no pseudocode required)

Name:

Let's assume we modify the 6.828 kernel to build a high-performance Web server. We add a driver for ethernet card in in the Web server environment. Interrupts from the card are directly delivered to the Web server's environment, and the interrupt handler executes on a separate stack in the Web server's environment (in a similar style as the 6.828 kernel sends page faults to environments.) The Web server's interrupt handler runs with interrupts disabled, processes the received interrupt completely (including any TCP/IP and HTTP processing), returns all resources associated with handling the interrupt, and re-enables interrupts. Web pages in the cache are served straight from the interrupt handler. CGI scripts and pages not in the cache are handled by the Web server on the main stack (not in the interrupt handler).

4. [10 points]: Does this implementation suffer from receive livelock? If so, sketch a sequence of events that will result in receive livelock. If not, explain why. (If you need to make any more assumptions about the architecture of the system, be sure to state them explicitly.)

III File systems and reliability

Modern Unixes support the `rename(char *from, char *to)` system call, which causes the link named "from" to be renamed as "to". Unix v6 does not have a system call for rename; instead, rename is an application that makes use of the `link` and `unlink` system calls.

5. [5 points]: Give an implementation of rename using the `link` and `unlink` system calls, and briefly justify your implementation.

```
int rename(char *from, char *to) {









}
```

Some editors use `rename` to make a new version of a file visible to user atomically. For example, an editor may copy "x.c" to "#x.c", make all changes to "#x.c", and when the user hits save, the editor calls `sync()` followed by `rename("#x.c", "x.c")`.

6. [5 points]: What are the possible outcomes of running `rename("#x.c", "x.c")` if the computer fails during the rename library call? Assume that both "#x.c" and "x.c" exist in the same directory but in different directory blocks before the call to `rename`.

Modern BSD UNIX implements `rename` as system call. The pseudocode for `rename` is as follows (assuming "to" and "from" are in different directory blocks):

```
int rename (char *from, char *to) {
   update dir block for ''to'' to point to ''from'''s inode // write block
   update dir block for ''from'' to free entry // write block
}
```

BSD's Fast File System (FFS) performs the two writes in `rename` synchronously (i.e., using `bwrite`).

7. [5 points]: What are the possible outcomes of running `rename("#x.c", "x.c")` if the computer fails during the rename system call? Assume that both "#x.c" and "x.c" exist in the same directory but in different directory blocks before the call to `rename`.

Assume the same scenario but now using FFS with soft updates.

8. [5 points]: How does using FFS with soft updates change this scenario?

Name:

IV Virtual machines

A virtual machine monitor can run a guest OS without requiring changes to the guest OS. To do so, virtual machine monitors must virtualize the instruction set of a processor. Consider implementing a monitor for the x86 architecture that runs directly on the physical hardware. This monitor must virtualize the x86 processor. Unfortunately, virtualizing the x86 instruction set is a challenge.

9. [5 points]: Using the instruction "mov %cs, %ax", give a code fragment that shows how a guest operating system could tell the difference between whether it is running in a virtual machine or directly on the processor, if the virtual machine monitor is not sufficiently careful about how it virtualizes the x86 architecture.

(List a sequence of x86 assembly instructions)

10. [5 points]: Describe a solution for how to virtualize the instruction "mov %cs, %ax" correctly. What changes do you need to make to the monitor?

(Give a brief description; no pseudocode required)

The monitor might emulate a load of CR3 as follows:

```
// addr is a physical address
void
emulate_lcr3(thiscpu, addr)
{
  Pte *fakepdir;

  thiscpu->cr3 = addr;
  fakepdir = lookup(addr, oldcr3cache);
  if (!fakepdir) {
    fakedir = page_alloc();
    store(oldcr3cache, addr, fakedir);
    // CODE MISSING:
    // May wish to scan through supplied page directory to see if
    // we have to fix up anything in particular.
    // Exact settings will depend on how we want to handle
    // problem cases.
  }
  asm("movl fakepdir,%cr3");
  // Must make sure our page fault handler is in sync with what we do here.
}
```

11. [15 points]: Describe a solution for handling a guest OS that executes instructions stored in its data segment (e.g., `user_icode` in the 6.828 kernel).

Name:

V Feedback

Since 6.828 is a new subject, we would appreciate receiving some feedback on how we are doing so that we can make corrections next year. (Any answer, except no answer, will receive full credit!)

12. [1 points]: Did you learn anything in 6.828, on a scale of 0 (nothing) to 10 (more than any other class)?

13. [1 points]: What was the best aspect of 6.828?

14. [1 points]: What was the worst aspect of 6.828?

15. [2 points]: If there is one thing that you would like to see changed in 6.828, what would it be?

16. [2 points]: How should the lab be changed?

17. [1 points]: How useful was the v6 case study, 0 (bad) to 10 (good)?

18. [1 points]: How useful were the papers, 0 (bad) to 10 (good)?

19. [2 points]: Which paper(s) should we definitely delete?

20. [2 points]: Which paper(s) should we definitely keep?

Name:

21. [1 points]: Rank TAs on a scale of 0 (bad) to 10 (good)?

22. [1 points]: Rank the professor on a scale of 0 (bad) to 10 (good)?

End of Quiz II

Name: