Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.828 Fall 2008

Quiz II

Most problems are open-ended questions. In order to receive credit you must answer the question as precisely as possible. You have 90 minutes to finish this quiz.

Write your name on this cover sheet.

Some questions may be 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 EXAM.

Please do not write in the boxes below.

I (xx/21)	II (xx/8)	III (xx/8)	IV (xx/8)	V (xx/8)
VI (xx/11)	**VII (xx/14)**	**VIII (xx/22)**	**Total (xx/100)**	

Name:

I File System Consistency

Ben is writing software that stores data in an xsyncfs file system (see the paper *Rethink the Sync* by Nightingale *et al*). He's nervous about crash recovery. To help himself test, he adds a new system call to his operating system called crash(), which powers off his computer without doing anything else. He also writes a simple "crash" command which calls his system call. His general test strategy is to perform some operations, call crash(), restart the computer and let the file system finish recovering, and then observe what files and data are on the disk.

1. **[7 points]:** First, Ben types the following commands to his shell:

```
% echo hello > foo
% crash
```

After the restart, is Ben guaranteed to see a file foo with contents "hello"? Why or why not?

2. [7 points]: Now Ben runs the following UNIX program which he believes is equivalent to the above commands:

```
pid = fork();
if(pid == 0){
  fd = creat("foo", 0666); // create a file foo
  write(fd, "hello\n", 6);
  close(fd);
  exit(0);
}
wait(&status); // wait for pid to exit
crash();
```

After the restart, is Ben guaranteed to see "hello" in foo? Why or why not?

3. [7 points]: What would the answers be for the xv6 file system? Why?

II Virtualization

The paper *A Comparison of Software and Hardware Techniques for x86 Virtualization* states that one of the goals of virtualization is "Fidelity: Software on the VMM executes identically to its execution on hardware, barring timing effects."

The paper also says, at the end of Section 3.2, that the software virtualization technique outlined in Section 3 does not translate guest user-mode code. That is, when the guest operating system executes instructions that would cause a switch to CPL=3 on real hardware, the VMM stops translating code and allows the original guest instructions to execute directly on the hardware.

4. [8 points]: Explain a way that a carefully constructed guest kernel and/or user-mode program could exploit direct execution of user-mode code to discover whether it was executing on a virtual machine. You should assume the system outlined in Section 3, running on a 32-bit x86.

III Fault Tolerance

Ben is impressed with the paper *Fault Tolerance Under UNIX*, by Borg *et al.* He builds a system that is identical – with one exception. In order to get better performance, his hardware has two busses, and every machine has a connection to both busses. When a machine needs to send a message, it selects one of the two busses at random, and sends the message on that bus.

5. [8 points]: Explain why this change will cause serious problems to the correctness of the system. Give a specific example of something that will likely go wrong.

IV Bugs as Deviant Behavior

Have a look at Figure 3 of the paper *Bugs as Deviant Behavior* by Engler *et al*. Suppose first function in Figure 3 looked like:

```
ssize_t proc_mpc_write(struct file *file,
                       const char *buff){
  retval = parse_qos(buff, incoming);
}
```

That is, suppose the first seven lines of the original function were deleted.

6. [8 points]: Would the checker described in Section 7.1 emit an error for this modified function? Why or why not?

V Livelock

Answer this question in the context of the paper *Eliminating Receive Livelock in an Interrupt-driven Kernel* by Mogul *et al.*

7. [8 points]: Figure 6-3 shows that performance with polling and no quota is quite poor under high input load, but polling with a quota performs much better. Explain what happens without a quota that results in almost zero performance under high input load, and how a quota helps solve this problem.

VI KeyKOS

Answer this question in the context of the paper *The KeyKOS Nanokernel Architecture* by Bomberger *et al*.

8. [4 points]: Suppose a machine running KeyNIX loses power while creating a new file in a directory. After the system is powered back up and executes to a quiescent state (e.g. so that any recovery code is executed), what are the possible states that the file system could be in? Circle all that apply (-1 point for each wrong answer).

A. The new file is allocated but the directory does not contain the new file.

B. The new file is not allocated but the directory contains a reference to the new (unallocated) file.

C. The new file is allocated and the directory contains a reference to the new file.

D. The new file is not allocated and the directory does not contain a reference to the new file.

9. [7 points]: Most file system implementations worry a great deal about what happens after a crash, but the KeyNIX file system has no explicit file system recovery code. Explain what strategy KeyNIX and KeyKOS use to recover from crashes such as the one above. In particular, would the KeyNIX file system recover to a consistent state if a new file was created but not added to its parent directory by the time the machine was powered off (and if so, how)?

VII HiStar

Answer this question in the context of the paper *Making Information Flow Explicit in HiStar* by Zeldovich *et al*.

Recall that KeyNIX maintains a global table mapping process IDs to capabilities (keys) that allow sending a message to that process's Unix keeper. When one process running on top of KeyNIX wants to send a signal to a different process, the sender's Unix keeper must check that the sender is allowed to send a signal to the recipient process (e.g. that both processes belong to the same user), before sending a "kill" message to the recipient's Unix keeper.

HiStar implements signals in a similar fashion: the Unix library locates the signal gate for the recipient process and sends a "kill" message to that gate. (Even though the Unix library has no global process ID table, it can still find the signal gate for a given process ID by enumerating that process's container to find the signal gate, since a process ID is the ID of that process's container object.)

10. [7 points]: The HiStar Unix library is not trusted; anything the Unix library can do, ordinary program code can also do. As a result, the Unix library cannot be trusted to check that signals are only sent to processes owned by the same user. How does HiStar prevent a malicious user program from sending signals to other users' processes?

11. [7 points]: Can KeyNIX be re-designed to prevent a compromised Unix keeper (controlled by an attacker) from sending signals to other users' processes? Sketch out a design that would run on KeyKOS, or describe why it would be difficult to implement using capabilities.

VIII RCU

Answer this question in the context of the paper *Read-Copy Update* by McKenney *et al*.

12. [7 points]: Illustrate a specific problem that could occur if the reference-counted linked list search code in Figure 4 did not obtain a read lock on list_lock. Be sure to indicate what would go wrong as a result.

13. [7 points]: The RCU version of the linked list search code in Figure 8 does not use locks. Describe the steps taken by RCU to ensure that the problem seen in the above question does not occur in this case.

14. [8 points]: Suppose you want to use RCU for a linked list in the xv6 kernel. Explain how to detect quiescence in xv6. To maximize read performance, you may not use any additional code (such as locks) around read accesses to the linked list.

End of Quiz

MIT OpenCourseWare
http://ocw.mit.edu

6.828 Operating System Engineering

Fall 2012

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.