

Systems Comprehensive Exam, Spring 2008

1 Instructions

This is a closed-book, closed-notes exam with a total of 100 points. You may not use any external source for answering these questions, including but not limited to the Internet, books, notes, or other people. Please direct any questions about this exam to Professor Patrick Bridges. Professor Bridges may be reached either in person in his office in 301B Farris, by phone at 277-3032 (office) or 314-4676 (cell), or by email at bridges@cs.unm.edu. Turn your exam in to Lynne Jacobson by 5:00 PM MDT on Monday, January 14, 2008. Exams *will not* be accepted after this time except by prior arrangement.

Type or write your answers to the stated number of questions in each of the following three sections. Make any *reasonable* assumptions necessary to answer the question, but be sure to state any assumptions that you make.

2 Short Answer - Answer 3 of 4 (30 points)

Briefly answer 3 of the following 4 questions. Your answer should be no longer than *one* paragraph.

1. The algorithm in Listing 1 claims to guarantee mutual exclusion. Does it? Explain how the algorithm works and how it guarantees mutual exclusion, or why it fails to do so.

The array `a[]` is shared and initially 0 in all locations. The variable `me` is set to each thread's ID ($0, 1, \dots, p - 1$). The variable `next` is `me + 1 % p`, where `p` is the number of participating threads.

In your explanation make sure to answer this question: Do this algorithm work for more than two processes?

Listing 1: "A Mutual Exclusion Algorithm?"

```
a[me] = me
while a[0] = me
    a[me] = 1
    while a[1] = next
        a[me] = 0
    endwhile
endwhile
critical section
a[me] = 0
```

2. Consider the code fragments in Listing 2 and 3. They are correct in a sequentially consistent system. Modern compilers and CPUs tend to reorder writes for performance reasons. Assuming both `flag` and `x` are initially 0, instruction reordering could cause Thread 2 to print "X is 0", instead of the expected "X is 12".

Show and explain how to use the *volatile* keyword to prevent reordering of the write statements.

Listing 2: "Thread 1"

```
x = 12;
flag = 1;
```

Listing 3: "Thread 2"

```
while (flag == 0);
printf("X is %d\n", x);
```

3. Leslie Lamport describes a **distributed** algorithm that allows us to order all observable events in a system (total order). Explain what is meant by a distributed algorithm. Provide a non-distributed algorithm that also provides a total ordering of events in a system. Just describe your algorithm; you do not need to provide actual code.

Why is it better to use a distributed algorithm (if possible)?

4. How do senders using Ethernet's CSMA/CD MAC protocol negotiate access to a shared link in case of a collision?

3 Medium Answer - Answer 2 of 3 (40 points)

Provide detailed answers to two of the following three questions. Be sure to state any assumptions you make and to fully justify your answers. Limit your answers to approximately one to two pages in length.

1. Deadlock

- (a) Write pseudocode for two threads of execution that can deadlock, using lock primitives like lock(x) and unlock(x) that can deadlock?
- (b) What conditions are necessary and sufficient to create deadlock on a concurrent program? How do the threads you described in part (a) of this question meet these conditions, either explicitly or through assumptions about the underlying runtime system?
- (c) How is deadlock traditionally avoided in concurrent systems, considering the conditions you described in your answer to part (b) of this question?

2. Journaling file systems

When a file system is mounted, the OS must check whether it is in a consistent state. If it is not in a consistent state, after a system crash for example, the file system must be repaired before it can be put in use. One of the key advantages of a journaling file system over the standard Unix file system is that it can be checked for consistency and, if necessary, repaired very quickly.

- (a) Show how a file system can get into an inconsistent state. Give a specific example, explain how it could occur, and what the consequences are.
- (b) Using your example from above, describe how a regular Unix file system would restore consistency. Again, be specific in describing what actions need to be taken and in what order.
- (c) Using the same example again, describe how a journaling file system would recover. Be sure to point out the differences to recovery of a regular Unix file system, and explain why a journaling file system excels in this area.

3. Memory Management

Virtual memory can provide:

1. Memory protection for multiple processes

2. The illusion of a larger amount of available physical memory in the system
3. Each process a flat view of its own memory.
 - (a) In one sentence, describe what role the TLB plays in efficiently implementing these goals.
 - (b) When switching between processes, we must flush the TLB if it contains no field for a “Process Identifier” or “Task Identifier”. Why?
 - (c) What are the limitations of a single-level page table with respect to goal 2? How does a multi-level page table help?
 - (d) In a couple of sentences, describe how an inverted page table could enable an extremely large virtual memory if an extreme form of goal 2 is desired.
 - (e) Consider the alternative to virtual memory of loading programs into different parts of a flat physical memory. Why is this a less desirable alternative? Ignore protection and focus on programability. (Hint: consider what addresses a program can use)

4 Design - Answer 1 of 2 (30 points)

Provide a *full* and *detailed* answer to one of the following two questions. Be sure to state any assumptions you make and to fully justify your answer.

4.1 Virtualization

The Intel architecture is not well suited for virtualization. For example, it is possible for a guest OS and even a user-level application to determine that it is running in a virtualized environment. The problem is that some sensitive instructions are not privileged and do not cause an exception. However, these instructions reveal information about the processor state that can be used by a user-level program to determine whether it is running above a guest OS, or the native OS of the machine.

One such instruction in the Intel architecture is SIDT “Store Interrupt Descriptor Table”. There is only one pointer to the interrupt descriptor table. Therefore, the virtual machine (VM) must set it so it points to its interrupt descriptor table, not the one of the guest OS. The location of the interrupt descriptor table for most VMs and guest OSs is well known, so it is easy to detect whether a program is running in a virtualized environment or not.

This is bad for researchers who set up so called honey pots to observe the behavior of malware. In order to contain malware in honey pots, they are run inside a VM. Increasingly, malware detects whether it is running in a VM and disables some of its worst features. This prevents researchers from observing the malware’s behavior when it is not running inside a VM.

Given the shortcomings of the Intel architecture in this regard, design a system software approach to solve this problem. For the purpose of this question, assume the following:

- There are more instructions than the SIDT instruction that can be exploited in a similar fashion.
- You are stuck with the CPU you are given (You cannot switch to an Intel VT or AMD V version of CPU that has enhanced virtualization support.)
- You cannot modify the guest OS, but you have complete information about how it works; i.e., you have access to the source code for inspection purposes.
- You are in complete control of the VM software

Be sure to state all of your assumptions. Describe in detail how your solution works and how it would be applied. Besides sensitive, unprivileged instructions, discuss other approaches to detect the presence of a VM and possible strategies to prevent them. List at least two.

4.2 Network Protocol Performance and Security

Recall that the TCP protocol uses a congestion control algorithm, where the sender adjusts the size of its congestion window (limiting sending speed) to control its utilization of network resources. Note that in TCP, the sender only uses the receipt or non-receipt of acknowledgements to infer the state of the network.

1. How might a receiver exploit the TCP congestion control algorithm to cause the sender to send more quickly than it should?
2. Describe a potential change to the TCP protocol (i.e. the bits that go on the wire and the associated sending and receiving code) that would prevent such an attack from working.
3. Describe a potential change to the TCP sending code *but not the receiver or wire protocol* that would allow the sender to detect a receiver misbehaving as described in part (a).